



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



*Institut für
Technische Informatik und
Kommunikationsnetze*

PTP Time Synchronization for FlockLab 2

Semester Thesis

Julian Huwyler

jhuwyler@student.ethz.ch

Computer Engineering and Networks Laboratory
Department of Information Technology and Electrical Engineering
ETH Zürich

Supervisors:

Roman Trüb

Reto Da Forno

Prof. Dr. Lothar Thiele

June 12, 2020

Abstract

Evaluation and development of wireless sensor network protocols often requires a test network for experiments. Testbeds help reducing the effort caused by deploying test networks, providing a reusable infrastructure, while also improving reproducibility of experiments. FlockLab is such a testbed, which consists of a system of distributed observers. In systems with distributed measurement collection, tight time synchronization is crucial. The new architecture of FlockLab, FlockLab 2, achieves the targeted accuracy of approximately 1 microseconds by synchronizing the time on each observer via a Global Navigation Satellite System (GNSS) receiver. This requires sufficient GNSS reception, which is often not available indoors. A protocol which can achieve sub-microsecond accuracy over the existing Ethernet infrastructure is Precision Time Protocol (PTP). In this thesis, PTP is configured on a BeagleBone single board computer with the Debian 9 operating system. In addition, the precision and accuracy in different setups was characterized. On the simplest setup, which consisted of two directly connected BeagleBones, an average offset of $0.003\mu\text{s}$ with a maximum offset of $3.070\mu\text{s}$ is measured. It is also seen that PTP is significantly influenced by high network load.

Contents

Abstract	i
1 Introduction	1
1.1 Motivation	1
1.2 Related Work	2
2 Background	4
2.1 Precision Time Protocol	4
2.1.1 Basic Elements and Hierarchy	4
2.1.2 Message Exchange and Delay Computation	5
2.1.3 Hardware Versus Software Timestamping	6
2.2 The Linux PTP Project	7
2.2.1 ptp4l	7
2.2.2 phc2sys	8
2.2.3 pmc	8
2.3 FlockLab	8
3 Materials and Methods	10
3.1 Time Synchronization on the BeagleBone	10
3.2 Measurement of Time Offset	12
3.2.1 Internal Measurement	12
3.2.2 External Measurement	13
3.3 Tools	13
3.3.1 Measurement Scripts	14
3.3.2 stress-ng	14
3.3.3 iperf	14
3.3.4 Raspberry Pi and Personal Computer	14
3.3.5 Switches	15

CONTENTS	iii
3.4 Measurement Setup	15
3.4.1 Direct Setup	15
3.4.2 One Switch Setup	17
3.4.3 Two Switches Setup	19
3.5 Data Post Processing	20
4 Results	22
4.1 Direct Setup	25
4.1.1 CPU Stress Test	26
4.1.2 Network Stress Test	26
4.2 One Switch Setup	27
4.3 Two Switches Setup	28
4.4 Measurements with Unexpected Artifacts	29
5 Discussion	31
6 Conclusions	32
6.1 Feasibility of Deployment	32
6.2 Future Work	32
A Appendix	1
A.1 Setup of Beaglebones	1
A.1.1 Configuration files	2
A.2 Automated Measurement Scripts	3
A.2.1 Internal Measurement Script	3
A.2.2 External Measurement Script	5
A.3 Data Post Processing	6
A.3.1 Quick Displaying of Collected Data [deprecated]	6
A.3.2 Overview of Data Processing	7
A.3.3 Overview of Files, Functions and Dependencies	8
A.4 Known Issues	15
A.5 Timetable	16
A.6 Original Project Assignment	17

Introduction

Nowadays, more and more devices are connected to the Internet with a rising amount of wireless devices [1]. Therefore, wireless protocols gain importance. For design and evaluation of wireless protocols a testbed helps tackling the challenges of repeatedly deploying test networks, as well as improving the reproducibility of experiments. FlockLab is such a testbed, providing distributed infrastructure to test wireless protocols. It is in service since 2012 and was recently renewed as FlockLab 2 platform. The testbed is currently extended by placing nodes on the rooftop to enlarge spacing between them, extending the existing short baseline distances.

In distributed networks, which obtain data in parallel, time synchronization is key to order events correctly. Therefore, tight time synchronization in the order of 1 microsecond is needed for FlockLab 2. Because of this requirement, the synchronization method was changed from a wireless protocol to GNSS synchronization. The GNSS module used in FlockLab 2 achieves 50ns accuracy, meeting the targeted 1 microsecond accuracy. If there is no sufficient GNSS reception available, the time is currently synchronized with the Network Time Protocol (NTP) achieving an accuracy in the order of milliseconds.

A protocol capable of achieving the targeted 1 microsecond accuracy via Ethernet is the Precision Time Protocol (PTP) as defined in the IEEE 1588 standard[2]. The goal of this thesis is to configure PTP in the FlockLab 2 environment and characterize the achievable time synchronization accuracy and quality.

1.1 Motivation

In the current setup we are restricted on the placement of FlockLab 2 observers, since we need sufficient GNSS signal reception, therefore we can only place them outdoors or near a window. This restriction is not desired for a testbed, as we want to measure indoor behaviour as well, thus requiring a different time synchronization method besides GNSS based time synchronization.

There are different methods to synchronize time, like building a dedicated system, but since the FlockLab 2 testbed already has Ethernet connectivity on every observer it is obvious to take advantage of this. One method would be to synchronize the observer via NTP, like it is on many devices, but this only provides millisecond accuracy. Another protocol is PTP which enables the targeted sub-microsecond accuracy. If we can show that the PTP protocol achieves this accuracy on the FlockLab 2 testbed as well, we can use it to complement the GNSS based synchronization.

1.2 Related Work

To give an idea of the achievable accuracy and precision with PTP, a summary of selected papers is given in this section.

A similar testbed to FlockLab 2 is Shepherd[3]. It uses SeeedStudio BeagleBone Green (BeagleBone) as its main platform as well and achieves tight time synchronization with GNSS and PTP. The performance achieved in a simple setup with one Ethernet switch states a time accuracy with a maximum offset of 2.4 μ s, while 91% of the measurements stay within an offset of 1 μ s. This offset is measured by sampling the falling edge of the SPI chip select(CS) signal on two nodes for 10,000 consecutive samples.

D.i.G. [4] is an out-of-band system developed for monitoring nodes in a high performance computing cluster. It is based on the BeagleBone Black, which uses the same processor and Ethernet hardware as the SeeedStudio BeagleBone Green (BeagleBone). In two papers by Antonio Libri et al. [5][6], they evaluate how NTP and PTP scale. They achieve an accuracy for PTP of 16.1ns and a precision of 513.7ns at best and the 99th-percentile is at 1.32 μ s. In their setup they used switches with PTP hardware support. As scalability is of importance in High performance Computing (HPC), the paper investigates this as well and finds that scalability for PTP is not considered an issue. The measurement of the time offset explained in [5], involved an oscilloscope to measure the delay caused by the Interrupt Service Routine(ISR). This allows to take the offset between ISR delay into account for the final computation of the system clock time offset. Hence, this improves the accuracy and precision of the measurements. Also it has to be noted that they did not synchronize their time to a precise absolute reference (e.g. GNSS) because it was not considered important in HPC.

In his master project[7] Mudassar Ahmed investigated the performance of PTP on the Beagle Bone Black in a simple setup for both software and hardware time stamping as well as under different conditions (Network, IO and CPU load). For software-based time stamping the average offset was between ± 0.5 ms, for hardware-based the accuracy reached approximately 50ns. It was found in this

paper, that CPU and I/O load did not influence the accuracy but network load did.

Background

In this section crucial components of this thesis are introduced and explained.

2.1 Precision Time Protocol

The Precision Time Protocol (PTP) is a protocol used to synchronize time on devices in a Local Area Network (LAN). It was first standardized in 2002 as the IEEE 1588 standard, extended in 2008 with the IEEE1588-2008 [2] also known as the PTP version two. Currently a third version is on its way, which takes the White Rabbit[8] extension into the standard and gives more flexibility in the configuration. In this thesis, the IEEE1588-2008 standard is used, because the *linuxptp* software, explained in section 2.2, uses this standard. In the following subsections, this standard is explained in detail.

2.1.1 Basic Elements and Hierarchy

PTP knows three different types of clocks: the boundary clock (BC), the BC and the transparent clock (TC). Every end node, which has only one PTP port (e.g. a client with only one Ethernet Port), is called an ordinary clock (OC). An OC can be either slave or master depending on the outcome of the execution of the best master clock algorithm (BMCA). This algorithm chooses the time source of the network, by finding the most precise clock based on the declared precision and other criteria of the clock. After the execution of this algorithm, the node with the best clock, according to the BMCA, acts as the master and therefore is the time source in the network. It is also known as grand master clock (GMC). Every other OC acts as a slave and synchronizes its time to the GMC.

Every node which connects OC devices or other nodes and supports PTP is either a BC or a TC. These special clocks are meant to reduce the jitter introduced by the network itself and lead to more precise synchronization. The

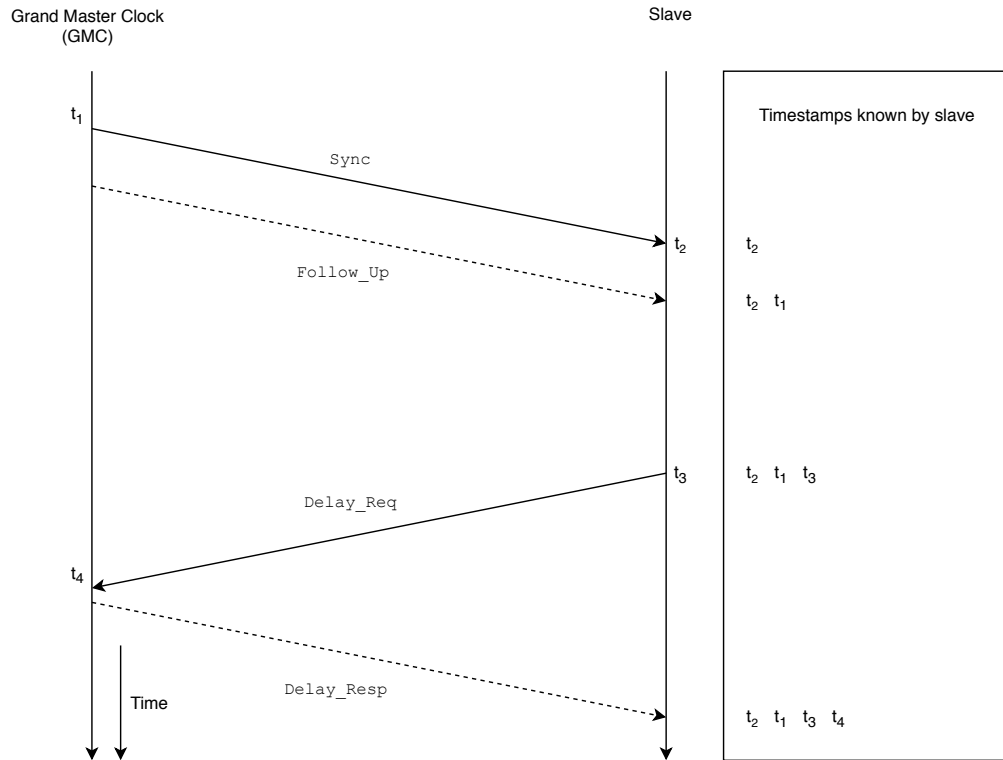


Figure 2.1: The delay request-response mechanism used by PTP. Solid arrows are Event Messages, dashed arrows are General Messages. This graphic was adapted from [6]

TC only measures the time spent in the device of a PTP packet and adds this to a specific message field in the packet. Therefore it is not a slave, nor a master and is called "transparent". The BC on the other hand has each port either configured in slave or master mode and actively synchronizes time with its peers. A node with multiple ports (e.g. a switch without PTP hardware support), which doesn't support PTP can significantly reduce the precision by introducing asynchronous path delays.

2.1.2 Message Exchange and Delay Computation

To be able to exchange data and measure time offset and delay, PTP uses two different types of messages, namely Event Messages and General Messages. The Event Messages are used for the synchronization and are a critical part of the PTP protocol, General Messages on the other hand are used to transmit timestamps or distribute announcement, management or signaling messages. A differ-

ence between the two message types is that the Event Messages get timestamped and any disturbance in the network on them will affect the quality of the synchronization.

Measurement of path delay and time offset is done with a delay response-request mechanism, which works as depicted in fig. 2.1: First, the master sends out a *Sync* message to all slaves connected to its port, while timestamping the time t_1 when the message was sent. The timestamp will be transmitted to the slaves right after in a *Follow-Up* message. On reception of the *Sync* message, the slave timestamps the message (t_2) and sends a *Delay_Req* message which is also timestamped (t_3) by the slave. The master will then timestamp the reception of this message (t_4) and send it back as a *Delay_Resp* message. After this the slave has all the data to calculate the path delay δ and the offset σ :

$$\delta = (t_4 - t_1) - (t_3 - t_2) \quad (2.1)$$

$$\sigma = \frac{t_2 - (t_1 + \frac{\delta}{2}) + t_3 - (t_4 - \frac{\delta}{2})}{2} = \frac{(t_2 - t_1) + (t_3 - t_4)}{2} \quad (2.2)$$

This calculation, adapted from [6], assumes a symmetric path delay, which explains why any disturbance in the network leading to asymmetric delay can significantly influence the quality of the PTP synchronization. The PTP maintains the synchronization by periodically performing this delay request response mechanism.

This request response mechanism has two configuration options: either end to end (E2E) or peer to peer (P2P). In the end to end mechanism the delay is calculated directly from GMC to slave. This allows for devices without PTP support to be present in the path. The peer to peer delay mechanism on the other hand calculates the delay for each segment of the path, therefore it requires PTP support on every node of the path.

2.1.3 Hardware Versus Software Timestamping

A lot of the jitter introduced in time synchronizations via Ethernet is caused from passing through the OSI layers. This is because the time when a packet physically arrives at the port of the device and the time the application gets notified is not deterministic. Therefore, we need timestamping as close to the physical layer as possible. In the PTP there are three options specified by the standard to timestamp the incoming and outgoing messages as seen in fig. 2.2:

A between the physical and mac layer of the OSI reference layer model

B in the kernel space or interrupt service routine

C at the application layer

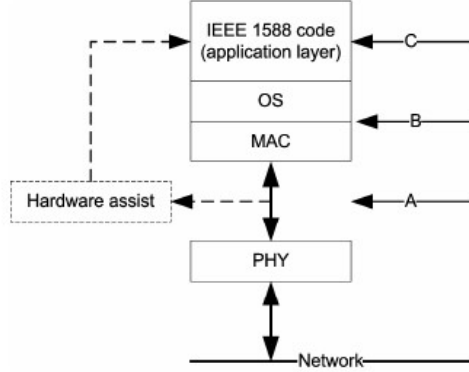


Figure 2.2: Possible Timestamping options from the PTP standard. Taken from [2]

The former two can be referred to as software timestamping whilst the latter requires hardware assistance. Therefore, we call it hardware timestamping. Hardware timestamps can be generated either between the MAC and PHY layer or even inside the PHY layer. Hardware timestamps have the advantage that they significantly reduce the delay and jitter introduced by the layers. In general the closer we capture a timestamp at the actual port, the more accurate the synchronization.

2.2 The Linux PTP Project

As the name suggests this is an implementation of the PTP for the Linux operating system. Included in this project are three software parts, namely *ptp4l*, *phc2sys* and *pmc*. The Linux PTP Project makes use of the PTP hardware clock (PHC) subsystem in Linux and has various configuration options required by the standard. Both software and hardware timestamping are offered via the Linux `SO_TIMESTAMPING` socket option. The tool synchronizes the system clock in two steps: First it synchronizes the PHC to a master clock in the network, then it synchronizes the PHC with the system time. In this thesis version 1.8 for the Debian 9 (Stretch) distribution is used.

In the following the Linux PTP components are explained in detail.

2.2.1 *ptp4l*

This software implements the time synchronization part of the PTP protocol. It synchronizes the PHC to a master clock in the network, or as a master clock synchronises the slaves in the network to its own PHC, according to the standard. If

the system is not capable of hardware timestamping and therefore does not have a PHC, the program directly synchronizes the system time. The configuration of the program can be adjusted via a configuration file.

2.2.2 phc2sys

For the second step in the synchronization, the program *phc2sys* synchronizes the system time to the PHC or vice versa. Various parameters can be adjusted via command line options.

2.2.3 pmc

The last piece of software is the implementation of the PTP management client as defined in the standard. It can be used to send and receive management messages, such as to get various measurements about the clock source in the network.

2.3 FlockLab

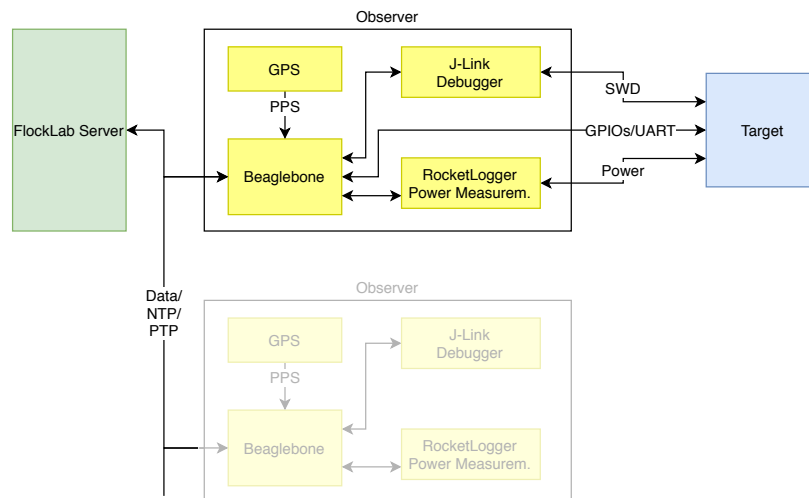


Figure 2.3: The FlockLab 2 Architecture. Taken from the original assignment in appendix A.6

FlockLab[9] is a testbed for wireless sensor networks, consisting of a network of FlockLab observers distributed on the campus of ETH Zurich, with a few remote nodes in the city. Each observer is directly connected to a device under test and has capabilities to control and trace inputs and outputs from this node

as well as to directly reprogram it.

In early 2020, a new version of FlockLab, known as FlockLab 2[10], was released. We can see a top level view on the architecture in fig. 2.3. Amongst more accurate power profiling based on the RocketLogger [11], Single Wire Debugging (SWD) and porting the services to a SoC single board computer platform, a GNSS module was added for tight time synchronization, as shown in fig. 2.3. This allows for precise sub-microsecond accuracy, which is required for high sampling rates in a distributed system.

The new FlockLab is designed as a cape for the BeagleBone Green single board computer. The advantage of the BeagleBone Green is, that it features an ARM 335x family processor, which has a 1 GHz single core processor and two Programmable Real-time Units (PRUs), providing the flexibility of an operating system combined with single cycle access to pins with the PRU [12]. Another advantage of the ARM 335x processor is its capability to support hardware timestamping for Ethernet packets, which enables PTP to use the hardware timestamping support for more precise time synchronization.

As an operating system the FlockLab 2 (FL2) platform currently uses Debian 9 (Stretch) with a Linux 4.14 kernel. This kernel supports the PHC and SO_TIMESTAMPING kernel functions which are required by the Linux PTP project to run in hardware timestamping mode.

Materials and Methods

In this chapter details are given about how the time synchronization works on the BeagleBone. Furthermore, the measurement setup for collecting precise timestamps and calculating the offset is explained. All the measurement setups which were executed are illustrated and a hypothesis on the performance is given for each one.

3.1 Time Synchronization on the BeagleBone

Section 2.1 stated that in a PTP setup we have one GMC with the rest of the OCs being slaves. To ensure that in our different measurement setups the BMCA always chooses the same device as a GMC, the configurations of *ptp4l* for the clock precision are set accordingly on each device. The device which has the most precise clock setting (and therefore gets elected as the GMC), synchronizes its time via a GNSS module to get the absolute time. This represents a realistic setup as it could be deployed in FlockLab 2, with a dedicated GMC and the observers synchronized via PTP. The whole time synchronization, depicted in fig. 3.1, is explained in the following.

As stated in section 2.3, in FlockLab 2 the time synchronization is implemented using a GNSS receiver. This receiver is located on the cape of the observer. Since we are only interested in the time synchronization, we do not need anything but this GNSS receiver from the cape. Therefore, to perform the measurements, we only use the BeagleBone itself combined with a ublox LEA-6 GNSS module. This also allows for direct access to the GPIO pins of the BeagleBone, which would be used up by the observer cape otherwise. The GNSS module has a USB connection and a direct outlet for the Pulse Per Second (PPS) signal. Via the USB connection we get the absolute time reference (UNIX timestamp) while the PPS is used to synchronize the BeagleBone's system time with high precision. In all the test setups, we only need one BeagleBone, which synchronizes its time with this GNSS module. Therefore, the GNSS module's USB port is connected to the GMC's USB port to provide power to the GNSS module

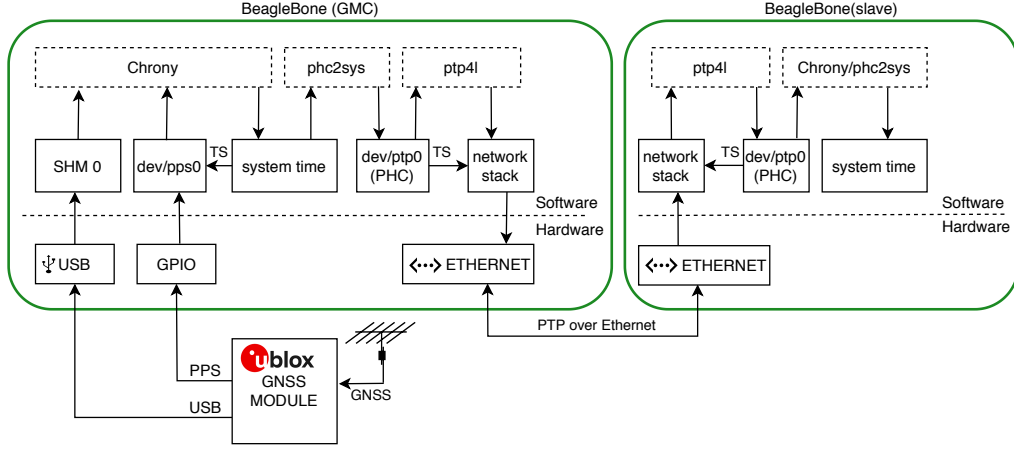


Figure 3.1: The complete synchronization of the time from a master to a slave.

and receive the absolute time reference, as seen in fig. 3.1. The PPS signal is connected to a GPIO pin, which can be configured as the input signal for a hardware timer in the AM335. The Linux kernel module *pps-gmtimer*[13] provides support for having precise PPS available in the Linux system as a pps device by timestamping the PPS signal with the system time. With the help of the *gpsd*[14] service we can receive the GNSS module's data from the USB port, which we use to provide the absolute time reference in the system via shared memory. These two modules are neglected in fig. 3.1 for the sake of simplicity. Finally, *chrony*[15] is used to synchronize the system time with the provided inputs. *chrony* can synchronize the system clock with various time sources, providing us flexibility to add backup sources if needed.

To configure a BeagleBone synchronized via GNSS as a GMC, the *linuxptp* project is used. In the first step, *phc2sys* synchronizes the PHC to the system time. In the next step, we use the software *ptp4l* to distribute the time from the PHC to our slaves in the network according to PTP. The Event Messages get timestamped with the time of the PHC as depicted in 3.1 with the arrow "TS". At the slave, *ptp4l* synchronises the PHC to the GMC according to PTP. Afterwards, either *chrony* or *phc2sys* can synchronize the system time to the PHC. In this thesis, *chrony* is mainly used due to the fact that it is more versatile and has already been used in FL2 before.

3.2 Measurement of Time Offset

One main challenge is to actually get precise timestamps from the system, since we cannot directly access any PPS signals. Two methods to get a timestamp from the system are proposed: One that logs values from the *pps-gmtimer* kernel module to a file (referred to as "Internal"). The other outputs a generated PPS signal, which is then measured with a logic analyzer (referred to as "External"). How these are exactly executed is the subject of this section.

3.2.1 Internal Measurement

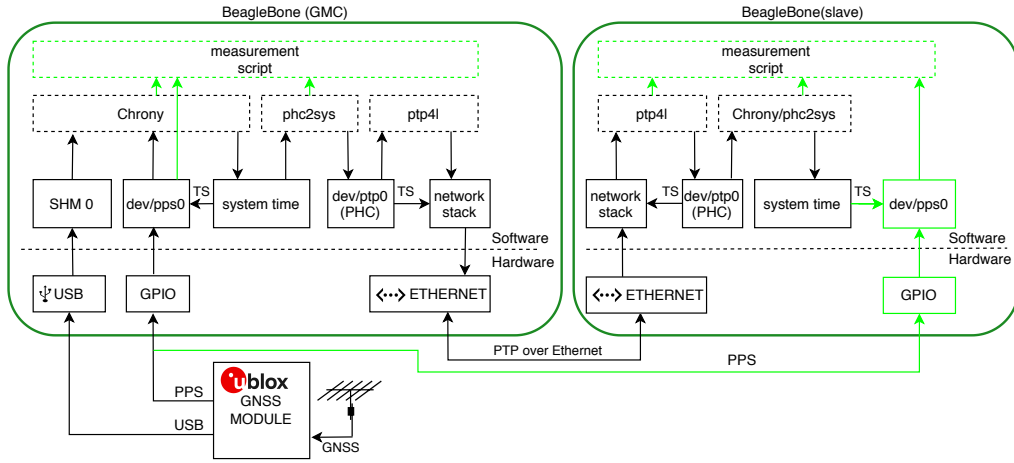


Figure 3.2: The Internal Measurement explained in the direct setup. The light green parts are added for the measurement of the time offset.

In this method we make use of the *pps-gmtimer*. By feeding the PPS signal from the GNSS module to the slave node as well, this module timestamps the PPS signal against the system time. This timestamp is a UNIX timestamp with nanoseconds, which we periodically collect (every second). To calculate the offset we simply take the decimals and if they are greater than 0.5 seconds we assume that the clock is slow and subtract 1 second from the offset giving us a negative offset which corresponds to a slow clock. This gives us the relative offset to the GNSS time source. We take this *pps-gmtimer* timestamp on the master as well to be able to see if the GNSS based synchronization of the master produces a significant offset.

Furthermore, the system log from the *phc2sys*, *ptp4l* and *chrony* software is collected. This allows for a deeper insight in the time synchronization chain, which enables more precise conclusions on where the offset error was produced.

3.2.2 External Measurement

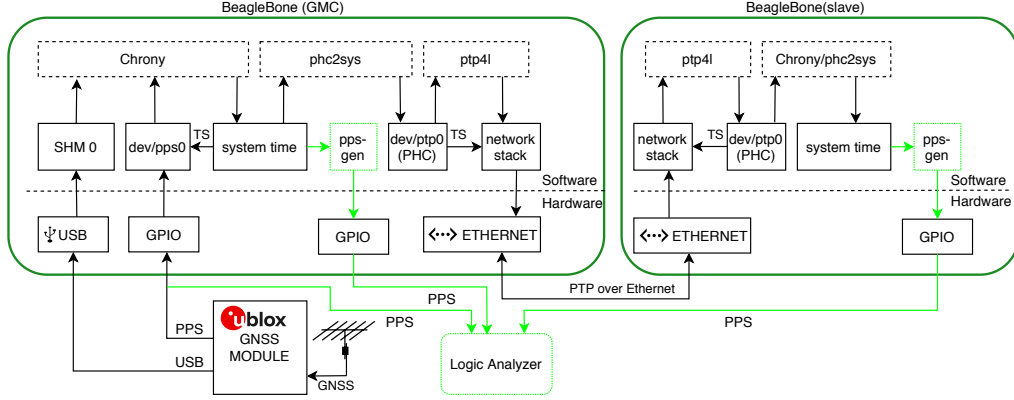


Figure 3.3: The external measurement explained in the direct setup. The light green parts are added for the measurement of the time offset.

The best case to measure the offset would be to measure the PPS signals produced by the various clocks directly. However, these signals are not physically accessible directly, but we have to output them somehow. To our best knowledge there exists no standard way to output a PPS signal in Linux. Therefore a kernel module was written to generate and output a PPS signal based on the Linux system time. This module, called *pps-gen*, outputs this PPS signal on one of the GPIO pins of the BeagleBone. To minimize interference, the *pps-gmtimer* module on the slaves is disabled while measuring externally. This achieves a precision in the range of microseconds because of timing variations in the execution of the kernel module. To collect these PPS signals, the Salae Logic 8[16] logic analyzer was used which sampled the PPS from the GPS module and all the generated PPS signals from the slaves.

In setups concerning network load, introduced in section 3.4.1, the external measurements were omitted, because it is expected that the effect of network load is best seen in the *ptp4l* data which is not accessible with external measurements.

3.3 Tools

Since we want to explore various aspects of the PTP and ensure the same conditions for all measurements, some software tools are needed. For this purpose, custom scripts to start the measurements and collect data were written. These scripts make use of two additional tools, to apply network and CPU load to the setup. All these tools are briefly introduced in this section.

3.3.1 Measurement Scripts

Two scripts were written, one for the external and one for the internal measurements. Both restart all the Linux system services which are used to synchronize the time at the beginning of each measurement, as well as inserting or removing kernel modules used in the corresponding measurement. Furthermore, the script timestamps the start and end of the measurement. These timestamps are used for the post processing of the data. The scripts are deployed to all Beagle-Bones and the start is initiated on the master device, which then distributes the command via SSH to the slaves automatically. The internal measurement script allows to generate artificial CPU and network load, to collect all the different measurement setups automatically. The external script does not have this ability, since the measurement with the logic analyzer software could not be started automatically at the same time and therefore this had to be done manually.

After a successful measurement, all the system logs, configurations and created documents get collected on each device and sent to the master. This way, collected data needs to be manually copied for data analysis only from a single device rather than from each device individually. All the data is later post processed as described in section 3.5.

For more information on the exact usage of these scripts please refer to the Appendix A.2.

3.3.2 stress-ng

For the application of artificial CPU load, the *stress-ng*[17] tool is used. The tool is either started automatically with the internal measurement script or manually for the external measurements.

3.3.3 iperf

The tool *iperf*[18] was used to apply artificial network traffic in the setups. In all of the setups UDP traffic was applied, because the used version (version 2) only supports bandwidth limitation for UDP traffic. UDP results in a asymmetric load on the network which was observed using the *netstat* tool.

3.3.4 Raspberry Pi and Personal Computer

A Raspberry Pi 4 Model B with Raspbian and a standard desktop Personal Computer(PC) running Ubuntu 18 with a *ASUS ROG STRIX Z270E GAMING* motherboard using a *Intel I219-V* Gigabit Ethernet controller were used additionally in the setups involving network load. In the cross traffic and link overload setups (explained in section 3.4.2 and section 3.4.3) the desktop was

used as an *iperf* server and the Raspberry Pi sent traffic to it. The Raspberry Pi was also used to generate traffic for the slave network overload setups described in section 3.4.2. Those two computers are used because they are able to produce 1 Gbit/s of traffic, while the BeagleBones's Ethernet port is only capable of 100 Mbit/s.

3.3.5 Switches

In the different measurement setups up to two network switches are used. The primary switch used is a *Planet GSD-805 Desktop Gigabit Ethernet Switch* [19] with eight ports each supporting 10/100/1000 Mbit/s Gigabit Ethernet. It also supports up to 16 Gbit/s non-blocking switch fabric. In this thesis it is referred to as switch 1. The other switch, referred to as switch 2, is an *Ubiquiti Unifi Switch 8 Model US-8* [20]. This switch supports 8 Gbit/s total non-blocking throughput and up to 16 Gbit/s switching capacity. Both switches do not support PTP.

3.4 Measurement Setup

In the following section the different measurement setups are described. For all setups *phc2sys* synchronizes the time 12 times a second and *ptp4l* performs synchronizes the time every second, which are the optimal synchronization rates according to [6]. Also only hardware timestamping is applied, since software timestamping misses the targeted accuracy of 1 microsecond by orders of magnitude [7]. For more information on the exact setup of the BeagleBone please refer to the Appendix A.1

All measurements are performed for a duration of 1000 seconds to collect 1000 data points. One slave is always configured as GMC and the others will act as slaves. In the measurement descriptions and results we refer to the GMC as *master* and to the different slaves as *slave24*, *slave25* and *slave26*. The numbers at the back of each slave name refer to the device name, which was *bb-24* for example, but this is only relevant for the static IPs and the distinction of the slaves in the results.

3.4.1 Direct Setup

In the direct setup, we connect two BeagleBones via an Ethernet cable as depicted in fig. 3.4. One has GNSS time synchronization and act as a GMC as explained in section 3.1, the other will therefore be a slave after executing the BMCA. This setup is used to get a benchmark on what is the best achievable time synchronization. Also, the different configurations are explored to evaluate if one performs better than the other. From these results a basic configuration

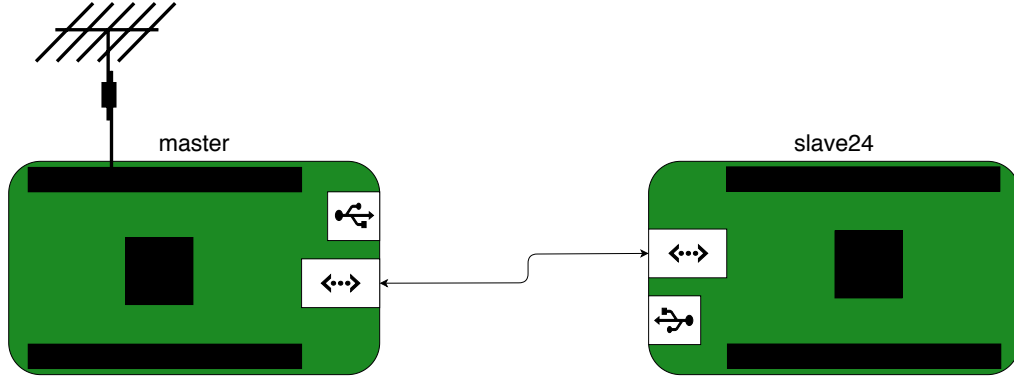


Figure 3.4: The Direct Setup with two BeagleBone. The antenna symbol indicates the time synchronisation via GNSS.

(abbreviated "BC" in the names of the measurements) is derived for the remaining measurements. To explain the names of the measurements (depicted in fig. 3.5) which will be executed, the BC is already explained here. The BC was chosen to be the end to end delay mechanism with the UDP over IPv4 transport for the reason explained in section 4.1. Despite testing different configurations of *ptp4l*, we also tested a different configuration of *chrony* and made one measurement with *phc2sys* running on the slave instead of *chrony*. The change in the configuration of *chrony* is the changed poll rate. It is set to a poll rate of 4, which corresponds to a 16 second interval. Therefore, *chrony* only takes the filtered measurements of 16 consecutive measurements.

It is expected that all the configurations perform equally well, since the precision of PTP should be achieved mainly through hardware timestamping rather than different configurations.

setup	delay mechanism	transport	slave poll rate	using phc2sys
P2P-L2	P2P	Layer 2 (IEEE 802.3)	0	no
E2E-UDPv4	E2E	UDP IPv4	0	no
BC-poll4	E2E	UDP IPv4	4	no
BC-phc2sys	E2E	UDP IPv4	0	yes

Figure 3.5: Names of the measurements in the direct setup. Note that *slave poll rate* is given as power of two.

CPU Stress Test

Since software is used to synchronize the time, it might get influenced by the CPU load, therefore CPU stress tests are performed using the *stress-ng* tool. The same setup as in the direct measurements is used and the basic configuration is applied. The *stress-ng* tool is started after a delay to give the system some time to stabilize the time synchronization and ensure that solely the applied CPU stress affects the time synchronization. In separate measurements, different percentages of artificial CPU load are applied to the CPU from the GMC as well as the CPU of the slave as seen in fig. 3.6.

In [7] CPU stress did not cause an issue with PTP, it is expected that the influence in these measurements should be minor as well.

setup	CPU load (master)	CPU load (slave)
BC-sCPU-50d	0%	50%
BC-sCPU-100d	0%	100%
BC-CPU-50d	50%	0%
BC-CPU-100d	100%	0%

Figure 3.6: Names of the measurements in the CPU stress setup. The lowercase d at the end of each name stands for "delay"

Network Stress Test

Network stress test are needed as PTP uses the network to synchronize time. The GMC is used as an *iperf* server and the slave sends UDP traffic to the the GMC. The bandwidth was limited with the `-b` flag to test certain loads. Measurements were performed for different network loads, as seen in fig. 3.7, to get an overview of how PTP performs. This covers the maximum network load for the BeagleBone, since it is only capable of sending and receiving of up to 100Mbit/s. UDP traffic was applied since the tool only allowed for UDP traffic bandwidth to be limited and our basic configuration is using UDP traffic as well. This measurement is expected to influence PTP because of the asymmetric network load. The asymmetry increases from about 1:10 to about 1:100'000 as the network load increases.

3.4.2 One Switch Setup

For measurements with more than one slave and simulating realistic scenarios, different measurements with devices all connected to one switch are performed as depicted in fig. 3.8. First, a measurement with only two BeagleBone connected to

setup	network load
BC-NW-10k	10 kbit/s
BC-NW-100k	100 kbit/s
BC-NW-1M	1 Mbit/s
BC-NW-10M	10 Mbit/s
BC-NW-100M	10 Mbit/s
BC-NW-200M	10 Mbit/s

Figure 3.7: Names of the measurements in the network stress setup.

the switch, one as GMC, the other as slave as in the direct setup, was performed. As a second measurement two more BeagleBone were connected to the switch and configured as slaves. After this, a measurement with cross traffic was performed, where we sent data through the switch from the RaspberryPi to the desktop PC. At last, network traffic was sent to a slave for the evaluation of the influence of traffic load on the slave separately. In the following the measurements are described in detail.

One Slave Connected to the Switch

In this setup we have the same setup as in the direct measurement, but the slave and master are connected via an Ethernet switch instead. With this measurement the goal is to be able to see the influence of the switch with nothing else influencing the measurement. It is expected, that the switch alone should not have an impact on performance in this setup, even if it has no hardware support of PTP, because the switch does not have any load applied to it except for the PTP messages. Therefore it should not have to delay packets. For all these measurements switch 1 was used as already stated in section 3.3.5.

Three Slaves Connected to the Switch

In this two more BeagleBones acting as PTP slaves are added to put the performance of PTP with multiple slaves to test. Since [5] stated that scaling up the number of PTP slaves should not be considered an issue, it is expected in this setup that three slaves will not have an influence on performance.

Cross Traffic Simulation

For the cross traffic simulation, the RaspberryPi and the desktop computer were used to exchange traffic with each other as described in section 3.3.4. With the produced network load on the switch it might be possible to reduce performance of PTP, because of the indeterminate behaviour of the switch in terms of

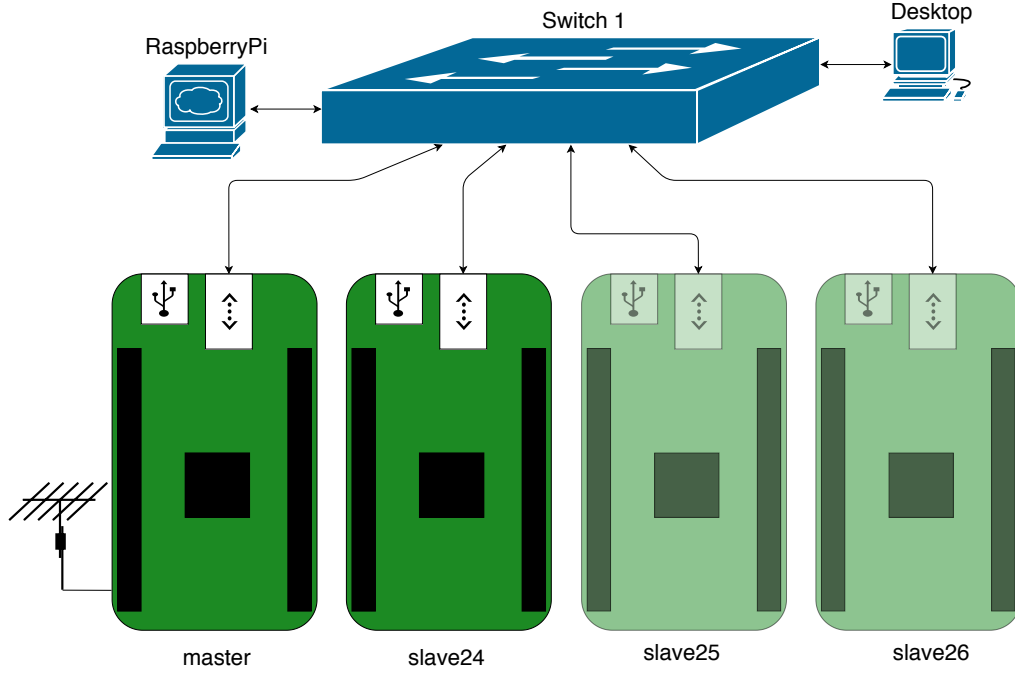


Figure 3.8: The Setup with one switch, one BeagleBone as a GMC and three BeagleBone as slaves. The slaves on the right are transparent because they are only used in certain measurements.

packet reordering at high rates. Although this is expected to be negligible, since the switch works way below its capacity limit of 16 Gbit/s switching traffic (see section 3.3.5 for more information on the switches).

Network Stress Test Slave

With this setup the goal is to get the influence of network load to a slave on the performance of PTP. The RaspberryPi is therefore configured to generate artificial network load to one of the slaves. A significant influence of this network load on PTP is expected since it can create asymmetric path delay in the network path.

3.4.3 Two Switches Setup

A common issue in networks is the overload of a link, often hindering packets to traverse this link and potentially slowing down the whole network. What effects an overloaded link has on the PTP is subject of this measurement. Therefore, we connect two switches with a single Ethernet cable (the link, which later is overloaded), as depicted in fig. 3.10. First, a measurement with three slaves

setup	number of slaves	network load (cross)	network load (slave)
BC-1slave	1	0	0
BC-3slave	3	0	0
BC-3s-cross	3	1 Gbit/s	0
BC-3s-slave1M	3	0	1 Mbit/s
BC-3s-slave5M	3	0	5 Mbit/s
BC-3s-slave10M	3	0	10 Mbit/s
BC-3s-slave100M	3	0	100 Mbit/s

Figure 3.9: Names of the measurements in the one switch setup.

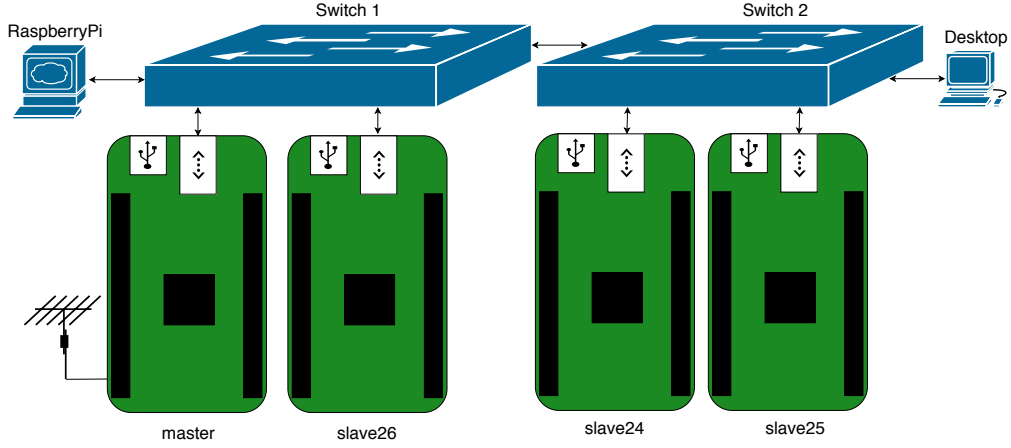


Figure 3.10: The setup with two switches, one BeagleBone as a GMC and three BeagleBone as slaves. Also traffic on the link can be applied via a RaspberryPi and another computer.

connected to different switches was performed to obtain a baseline measurement on how accurate the synchronization is without the link overload effect. The additional switch is expected to not have an influence on the performance of the PTP. Secondly, we use the RaspberryPi connected to one switch and the desktop computer connected to the other switch, to overload the 1 Gbit link with 1 Gbit/s traffic. Here it is expected to influence the performance because of the large asymmetric link load which causes an asymmetric path delay for the slaves connected to the switch 2 without the GMC present.

3.5 Data Post Processing

After the measurements are collected, further processing is required to be able to display and analyze the data. It was post processed using Python with the pack-

setup	network load on link
BC-2switch	0
BC-link-1M	1 Mbit/s
BC-link-10M	10 Mbit/s
BC-link-100M	100 Mbit/s
BC-link-1G	1 Gbit/s

Figure 3.11: Names of the measurements in the two switch setup.

ages *pandas*, *bokeh* and *matplotlib*. Everything was set to the master timescale. This allows for consistent and comparable measurement points. Furthermore, the first 200 seconds of each measurement were dropped to get rid of the unstable initialization part of the measurements. Afterwards, different summaries of all the setups and measurements as well as plots were made using this processed data. For more details please refer to the documentation of the code in Appendix A.3.

Results

In this section the results of the described measurements are presented in the corresponding sections. A complete summary with mean and maximum offset of the devices is given in Table 4.1 for the internal and in Table 4.2 for the external measurements, which are both found in the appendix.

setup	device	master		slave24		slave25		slave26	
	measure measurement	avg [μ s]	min/max [μ s]	avg [μ s]	min/max [μ s]	avg [μ s]	min/max [μ s]	avg [μ s]	min/max [μ s]
direct	P2P-L2	-0.196	2.690	-0.307	-1.015	NaN	NaN	NaN	NaN
	E2E-UDPv4	0.015	2.758	0.003	3.070	NaN	NaN	NaN	NaN
	BC-poll4	-0.307	2.311	-0.582	1.676	NaN	NaN	NaN	NaN
NW	BC-phc2sys	-0.025	3.023	0.048	2.320	NaN	NaN	NaN	NaN
	BC-NW-10k	-0.185	3.040	-0.303	2.551	NaN	NaN	NaN	NaN
	BC-NW-100k	-0.192	2.174	-0.314	-1.470	NaN	NaN	NaN	NaN
	BC-NW-1m	-0.209	2.674	-0.377	-1.231	NaN	NaN	NaN	NaN
	BC-NW-10m	-0.132	3.232	-0.532	-2.742	NaN	NaN	NaN	NaN
	BC-NW-100m	-0.676	-3.820	-413.606	-828.926	NaN	NaN	NaN	NaN
	BC-NW-200m	-0.207	-1.569	37.427	42.701	NaN	NaN	NaN	NaN
	BC-sCPU-50d	-0.049	2.814	0.208	2.843	NaN	NaN	NaN	NaN
CPU	BC-sCPU-100d	-0.038	2.886	0.614	2.616	NaN	NaN	NaN	NaN
	BC-CPU-50d	-0.615	-7.914	-1.525	-9.869	NaN	NaN	NaN	NaN
	BC-CPU-100d	-1.587	-12.307	-3.202	-13.708	NaN	NaN	NaN	NaN
	BC-2switch	-0.005	3.127	0.153	2.452	1.842	3.261	1.878	3.302
2-switch	BC-link-1M	0.274	3.149	0.048	-1.233	2.150	3.874	1.725	3.314
	BC-link-10M	-0.188	2.473	-0.490	-3.223	1.621	3.053	1.254	2.823
	BC-link-100M	-0.032	2.792	-0.235	2.695	1.723	3.302	1.318	3.341
	BC-link-1G	-0.267	3.072	1.103	4.332	1.496	3.206	2.844	6.817
1-switch	BC-1slave	-0.048	2.755	-0.252	2.512	NaN	NaN	NaN	NaN
	BC-3slave	0.009	2.585	0.079	3.742	1.995	4.092	2.086	3.783
	BC-3s-cross	-0.103	3.063	-0.153	1.674	1.678	3.254	1.745	3.168
	BC-3s-slave1M	0.002	3.193	-1.228	-63.947	1.901	3.521	1.850	3.423
	BC-3s-slave5M	-0.042	2.888	-5.725	-103.984	1.609	3.185	1.462	3.066
	BC-3s-slave10M	-0.087	1.921	-7.226	-127.537	1.893	3.239	1.809	3.418
	BC-3s-slave100M	0.071	3.399	-5815.683	15027.272	1.776	3.233	1.720	3.192

Figure 4.1: All setups with the corresponding measurements from the internal measurements and their mean and maximum of the *pps-gmtimer* offset given in microseconds

setup	device measure measurement	master		slave24		slave25		slave26	
		avg [μ s]	min/max [μ s]	avg [μ s]	min/max [μ s]	avg [μ s]	min/max [μ s]	avg [μ s]	min/max [μ s]
direct	P2P-L2	-2.416	-4.900	-2.390	-7.280	NaN	NaN	NaN	NaN
	E2E-UDpv4	-2.520	-5.580	-2.431	-4.400	NaN	NaN	NaN	NaN
	BC-poll4	-2.591	-6.200	-2.366	-4.250	NaN	NaN	NaN	NaN
	BC-phc2sys	-2.565	-7.159	-2.429	-5.980	NaN	NaN	NaN	NaN
CPU	BC-sCPU-50d	-2.621	-5.600	-4.342	-9.800	NaN	NaN	NaN	NaN
	BC-sCPU-100d	-2.550	-5.075	-6.243	-9.876	NaN	NaN	NaN	NaN
	BC-CPU-50d	-3.670	-9.725	-1.425	8.400	NaN	NaN	NaN	NaN
	BC-CPU-100d	-3.933	16.725	0.400	20.475	NaN	NaN	NaN	NaN
2-switch	BC-2switch	-2.708	-5.125	-2.696	-9.375	-4.705	-11.950	-4.933	-12.675
1-switch	BC-3slave	-2.833	-6.775	-2.554	-6.249	-4.941	-12.399	-5.013	-12.825
	BC-3s-cross	-2.857	-6.201	-2.523	-5.000	-4.680	-12.000	-4.795	-12.425
	BC-1slave	-2.682	-5.320	-2.558	-5.320	NaN	NaN	NaN	NaN

Figure 4.2: All setups with the corresponding measurements from the external measurements and their mean and maximum of the offset given in microseconds

4.1 Direct Setup

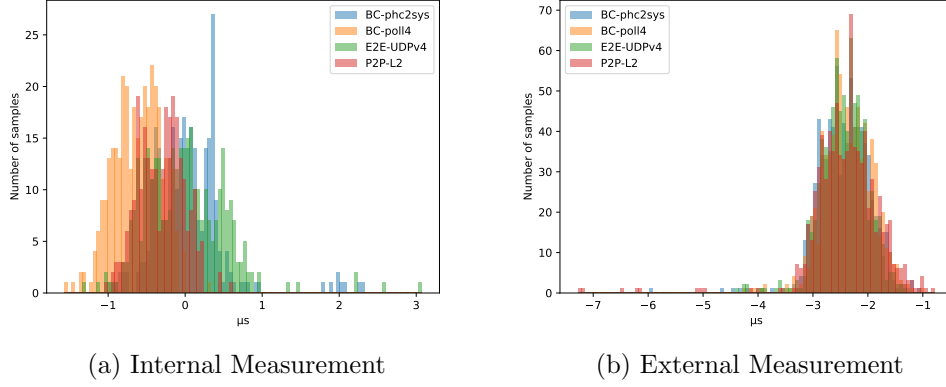


Figure 4.3: Distribution of offset on the slave for each tested configuration.

In the direct setup only a subset of configurations for *ptp4l* worked, namely the peer to peer (P2P) option combined with the layer 2 transport(L2) and the end to end option combined with UDP transport on IP version 4 (UDpv4). UDP on IP version 6 did not work at all. If a configuration worked or not was determined by the slave clock being able to recognize the GMC as a master. A full overview on the working configurations is given in fig. 4.4

These two different setups do not differ significantly in terms of offset. The time

	L2	UDpv4	UDpv6
E2E	no	yes	no
P2P	yes	no	no

Figure 4.4: A matrix of the combinations of configurations that worked in the direct setup.

offset on the slave of the E2E-UDpv4 configuration has 97% of its values inside the $\pm 1\mu s$ boundary. Since we use network hardware that does not support PTP in our tests, the E2E-UDpv4 setup was defined as the basic configuration (BC) for the remaining measurements. Furthermore, it can not be assumed that the network has PTP support on every node in the FlockLab 2 network.

The changed poll rate of *chrony* did not influence the precision or accuracy significantly. If the time synchronization on the slave was done with *phc2sys* instead of *chrony* the precision and accuracy did not change significantly as seen in Fig. 4.3.

The distribution of the BeagleBone which acted as a GMC was similar to the one from the slave.

In Fig 4.3b we see that the external measurement has an offset from zero. This

is likely caused by the non-deterministic execution of the kernel module.

4.1.1 CPU Stress Test

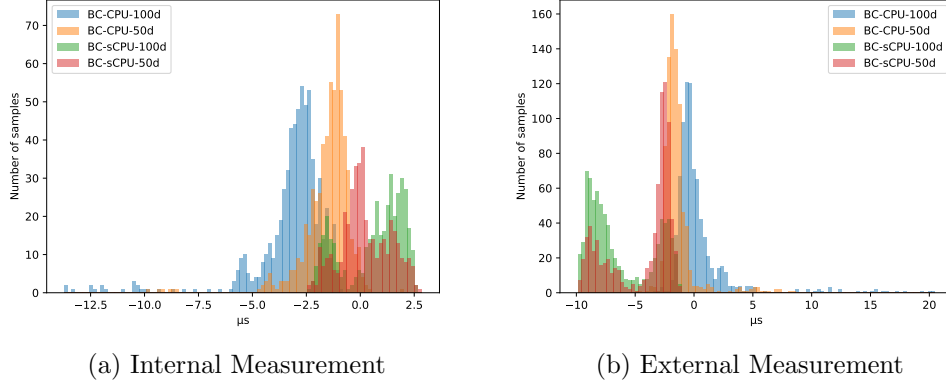


Figure 4.5: Applying CPU load to master or slave has an impact on the PTP performance

If we apply CPU load to the master, the performance of the time synchronization is influenced, as depicted in Fig. 4.5. The maximum offset in the measurement with 100% CPU load on the master leads to an offset of $12\mu\text{s}$ and an average of $0.3\mu\text{s}$ on the master. If we instead apply CPU stress to the slave, the performance is impacted as well, but less significant as with the master CPU-load. We get an average offset of $0.6\mu\text{s}$ and a maximum offset of $2.6\mu\text{s}$ at 100% CPU load on the slave for the internal measurements. For the external measurements, shown in Figure 4.5b, the maximum offset increases to $9.876\mu\text{s}$ with an average of $-6.243\mu\text{s}$.

4.1.2 Network Stress Test

Already starting at a network load of 10 Mbits/s, the time synchronization on the slave suffers as shown in Fig 4.6. It has to be noted, that the application of network load using *iperf* applied a CPU load of up to about 30% on the slave, therefore this might also have an influence. The effect of the network load can be seen in the offset of the *ptp4l* measurements depicted in Fig. 4.6b

For the 100 Mbits/s network load measurement seen in Fig. 4.7 we get nearly an uniform distribution of the offsets. Also, for this measurement we do not get any *ptp4l* values from the slave since it could not receive the packets to calculate the offset. The log from *ptp4l* shows more delay timeouts for higher network load.

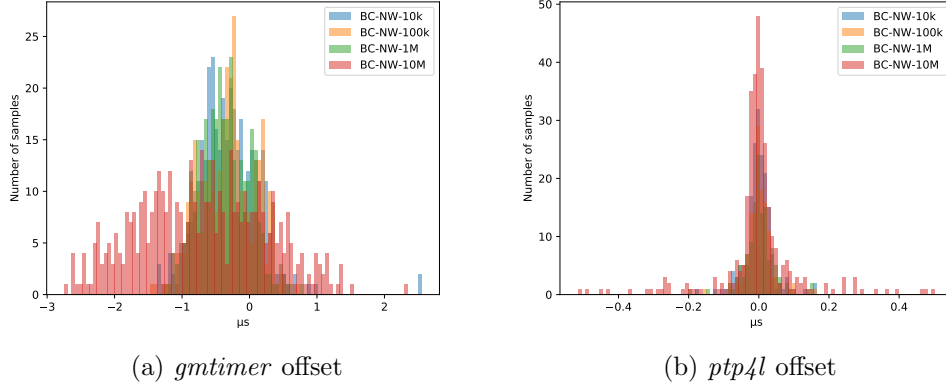


Figure 4.6: It is seen that the distribution changes already at 10 Mbit/s of network load.

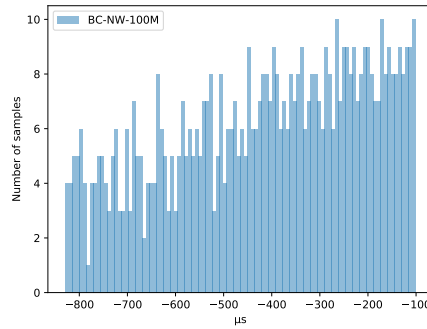


Figure 4.7: If the slave sends 100 Mbit/s network load to the master, we approximate a uniform distribution.

4.2 One Switch Setup

By adding a switch in the setup and adding more slaves to it, the performance did not change significantly. Even with 1 Gigabit/s cross traffic the performance did not suffer. We only see larger maximum offsets in Fig. 4.8, compared to the direct setup. Also the external measurements did not get influenced by these parameters as well, depicted in Fig. 4.8b.

Figure 4.9 shows us the behaviour of the slave with different network loads on the distribution of the *ptp4l* data. It becomes apparent that at a network load of 10 Mbits/s leads to a more widespread distribution on the slave with a mean of $-7.2\mu\text{s}$ and a maximum offset of $125\mu\text{s}$ for the *gmtimer* measurement. At 100 Mbit/s (Fig. 4.9b) these numbers change to a mean of $-5815.7\mu\text{s}$ and maximum offset of $15027.3\mu\text{s}$ as depicted in fig. 4.1.

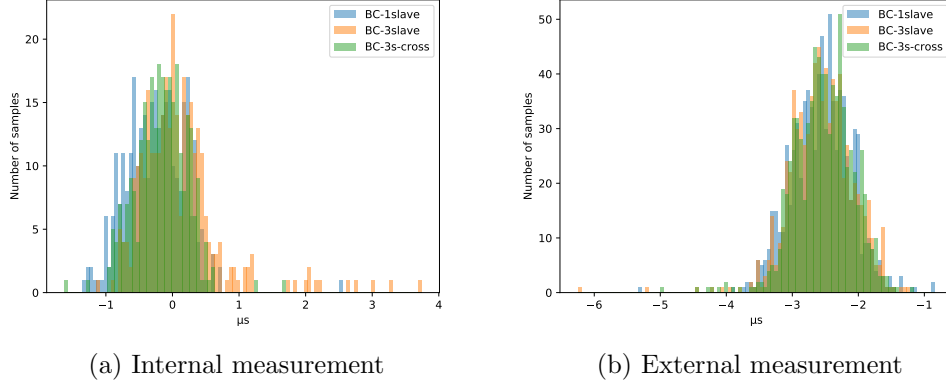
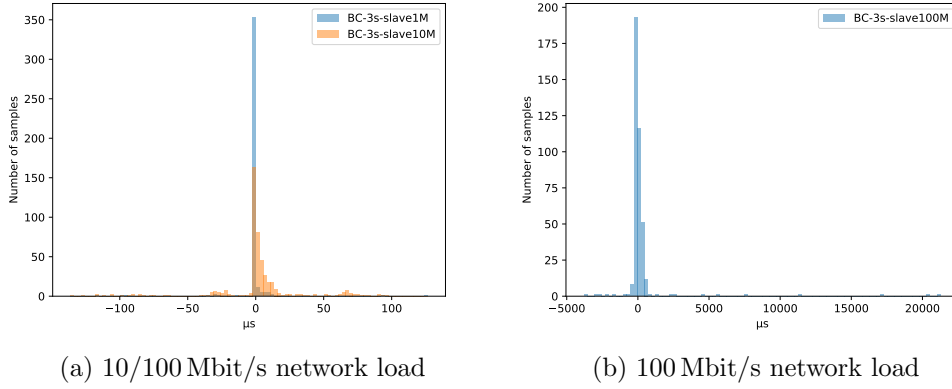


Figure 4.8: The measurements from the slave24 in the one switch setup

Figure 4.9: Already a small asymmetric network load of 10 Mbit/s can influence the performance of PTP significantly as seen in the distributions of the *ptp4l* calculated offsets

4.3 Two Switches Setup

For the setup with two switches, the measurement with three slaves led to similar results as in the direct setup with an average of $0.15\mu\text{s}$ and a maximum offset $2.45\mu\text{s}$. Although, the external measurements had outliers with a larger offset as in the direct setup as seen in Fig 4.10. When load is applied to the link between the two switches the performance is not influenced significantly except for the measurement with 1 Gbit/s load as depicted in Fig. 4.11. There the mean was $1.1\mu\text{s}$ with a maximum offset $4.3\mu\text{s}$. We see in Fig. 4.11b that the offset calculated by *ptp4l* shows a wider distribution for the 1 Gbit/s measurement as well.

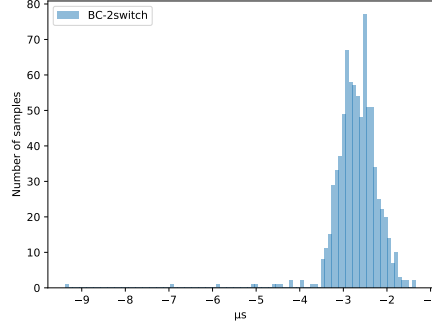


Figure 4.10: The distribution of the external measurement without any network load on the link showed larger maximum offsets.

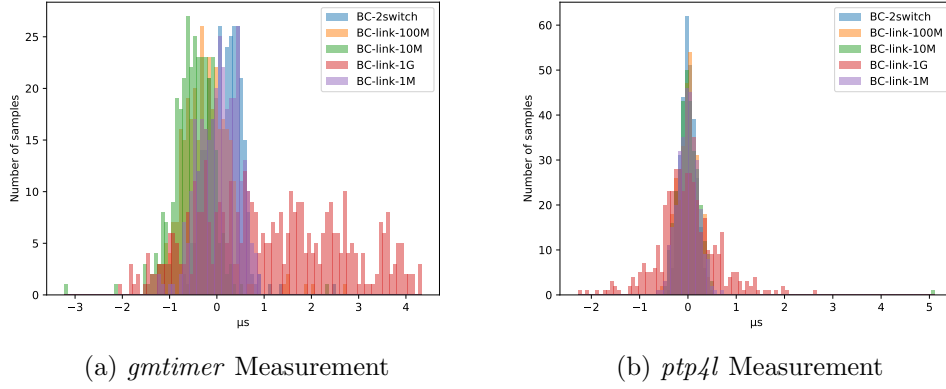
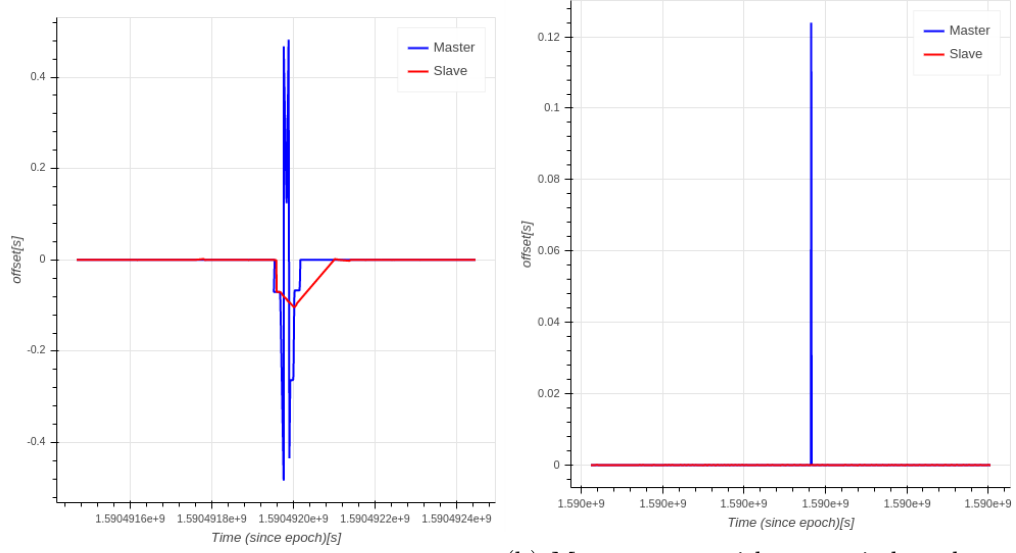


Figure 4.11: A link overload can influence performance of *ptp4l*

4.4 Measurements with Unexpected Artifacts

Every now and then, artifacts were discovered in the data with a large offset (approximately 0.1 up to 0.5 seconds) on the master, depicted in Fig. 4.12. It is unknown where this error originates.



(a) Measurement with link load of 50 Mbit/s (b) Measurement with one switch and one slave.

Figure 4.12: Two time series with artifacts.

Discussion

In the direct setup the data suggests, that the performance of each configuration is similar, which was expected.

In the measurements with artificial CPU load we verified, that the CPU load on either slave or master can influence the performance. Considering the values of *ptp4l* it can be concluded, that this influence does not originate from PTP. It is supposed that a user space program suffers from high CPU load, which would draw the attention to *chrony*, but this speculation is not proven in this thesis.

Applying network load had a noticeable impact on the performance. This behaviour was expected as well because of the assumption of symmetry in the calculation of the offset in PTP. The asymmetric network load from the slave to the master is therefore sub optimal for the performance of PTP. However, regarding the CPU load that the network load simulation causes, this could be influencing the results.

After examining the performance of PTP on one switch and on a setup with two switches, it is apparent, that adding three slaves into the network has a negligible impact on the performance. However, we cannot make statements about setups with large numbers of slaves, as we have not measured with more than three slaves. Although, according to [5] scaling up the number of slaves should not be an issue.

Adding one or two switches has a negligible impact as well. The cross traffic on the switch itself proved to not have a significant impact on the performance as well. With overloading the link between two switches, the performance dropped as expected.

Conclusions

During this semester thesis, a PTP configuration for the BeagleBone was successfully found. Through different measurements the performance of this protocol was tested on different setups to specify its accuracy and precision. Nearly all the expectations were met, except for the CPU load measurement, which showed an unexpected impact. The goal of sub microsecond accuracy and precision could only be met in setups with no network or CPU activity.

6.1 Feasibility of Deployment

The question remaining is whether PTP can be deployed to FlockLab 2. From the conclusion in the previous section it can be concluded that attention has to be paid to CPU and network load. Furthermore, a question that remains open, is if the protocol scales to bigger networks without hardware support of PTP. Considering that only under perfect conditions the targeted 1 microsecond accuracy and precision could be met, it is unclear if in a real network setup this target can be achieved. Also, a dedicated GMC should be deployed in the network to ensure that the CPU load on the master stays as low as possible.

6.2 Future Work

After examining the network load impact on PTP using UDP traffic to the slave, it is obvious to perform measurements in the other direction. Despite the fact that we already have direction from the slave to the master in the direct setup, the isolated impact of traffic going to the slave has not been evaluated yet.

Long time measurements (e.g. during 24h) would be interesting too, to determine the long term stability of the system.

It is also apparent to do a measurement on the FlockLab 2 itself. With this the performance could be evaluated and a status quo in terms of accuracy and precision can be defined.

Appendix

A.1 Setup of Beaglebones

In this section the whole process of setting up a BeagleBone as a FL2 observer with PTP capabilities is described. First a step by step solution is presented and in the further subsections the configurations are explained.

1. Setup observer according to FL2 wiki¹
2. Install the *linuxptp* project with `apt install linuxptp`
3. Create a new folder under `/etc/systemd/system` which is named `phc2sys.service.d`, which is used to place the configuration of the `phc2sys` file
4. Take the configuration files and place them at the following locations:
`chrony.conf` → `/etc/chrony/chrony.conf`
`phc2sys.conf` → `/etc/systemd/system/phc2sys.service.d/phc2sys.conf`
`ptp4l.conf` → `/etc/linuxptp/ptp4l.conf`
5. enable the *ptp4l* and the *phc2sys* service
6. restart the services *chrony*, *phc2sys* and *ptp4l*

This is the same for the setup of an observer as a PTP master or a slave, except the configuration files are different. If *phc2sys* is not used on the slave, the corresponding steps for it can be skipped. Also if the master only serves as a GMC, the setup of the whole observer can be skipped and the tools for the GNSS time synchronization can be added manually instead.

¹<https://gitlab.ethz.ch/tec/research/flocklab/flocklab2/-/wikis/home>

A.1.1 Configuration files

chrony

Two different configuration files are provided: One for master and one for slaves. On the master we take the existing file from the FlockLab 2 observer and only change the option `makestep` to `makestep 1 -1` such that the clock gets stepped if the offset is bigger than 1 second.

On the slave we define one `refclock` as `refclock PHC /dev/ptp0 refid PTP poll 0 offset -37` which lets chrony synchronize the time to the PHC.

phc2sys

To be able to pass configuration values to *phc2sys* the drop-in replacement method from *systemd* is used, therefore we have to replace the execution option as done in the file.

On the master the source is set to the systemtime and the PHC as slaveclock. The synchronization rate is set to 12 Hz and an offset of 37 seconds is set for the right conversion from UTC(Universal Time Coordinated) to TAI(International Atomic Time). The summary option `-u` is set to 12 measurements to log 1 measurement to the systemlog per second.

On the slave the systemtime is set as slaveclock and the PHC as source. Everything else is the same as in the master, except for the `-w` flag is set to wait until *ptp4l* is synchronized.

ptp4l

In both configuration files we set the device to the corresponding delay mechanism and network transport mode of a measurement. In the basic setup this would be end-to-end (E2E) and UDPv4 transport mode. The synchronization interval is set to 0, which corresponds to a 1 second interval and the announce interval is left at its default value of 1. Because we use hardware timestamping and the announce and synchronization interval is similar we set the timestamp processing mode `tsproc` to `raw`, which should perform well according to the Debian manual². Delay filter and length are left at default values.

For the master we configure the values in the `[global]` section to reflect the GNSS clock we are using, therefore setting accuracy to 0x20 and the `clockclass` to 6. On the slave we use default values but set the `step_threshold` to 1.0 seconds to let the program step the clock if it has a bigger offset than 1 second.

A file named `ptp4l-backup.conf` is provided where all values are left like it was after installing.

²<https://manpages.debian.org/testing/linuxptp/ptp4l.8.en.html>

A.2 Automated Measurement Scripts

In this section the internal and external measurement scripts and their usage are described. Both need superuser permissions to work since they need to restart applications. Also the devices are assumed to use fixed ip addresses in the format `192.168.1.bb` where `bb` is the number of the BeagleBone.

A.2.1 Internal Measurement Script

This script is used to automatically start measurements for this thesis. It is meant to be started on the PTP master with the `-m` master flag.

Prerequisites

For the CPU stresstests the packet *stress-ng* has to be installed on all devices. In network related measurements the tool *iperf* has to be installed on all devices

Synopsis

```
./startMeasurements -tmp [-n number-of-measurements] [-f  
foldername] [-l slave-nw-load] [-x cross-traffic] [-c cpu-stress] [-s  
slave-cpu-stress] [-b beaglebone] [-d delay]
```

Options

<code>-n <i>number-of-measurements</i></code>	the number of measurement steps in seconds
<code>-t</code>	give the time in minutes instead of the number of measurements. The number of measurements given with <code>-n</code> will be multiplied by 60.
<code>-m</code>	this option has to be set if the script is started on the master. It will allow it to start the same options given by other flags on other BeagleBone specified with the <code>-b</code> flag.
<code>-p</code>	use <i>phc2sys</i> on the slaves as time synchronization instead of <i>chrony</i> .
<code>-f <i>foldername</i></code>	the name of the folder where all the data produced gets saved to. Default name is <i>measurements</i>
<code>-l <i>slave-nw-load</i></code>	applies the specified network load to the BeagleBone 24 (ip 192.168.1.24). The value should be the same as with the bandwidth flag in the tool <i>iperf</i> e.g. "1M" for 1 megabit/s. <i>iperf</i> has to be running on the BeagleBone 24 as a server for UDP traffic (<i>iperf -usD</i>)
<code>-x <i>cross-traffic</i></code>	applies the specified network load from a RaspberryPi to a host with ip address 192.168.1.148 for simulation of crosstraffic. <i>iperf</i> has to be running on the host as a server for UDP traffic (<i>iperf -usD</i>)
<code>-c <i>cpu-stress</i></code>	applies the specified percentage of load to all CPU cores of the master BeagleBone using <i>stress-ng</i>
<code>-s <i>slave-cpu-stress</i></code>	applies the specified percentage of load to all CPU cores of the slave BeagleBone 24 using <i>stress-ng</i>
<code>-b <i>beaglebone</i></code>	specify the ID of the BeagleBone to be used in this measurement, e.g <code>-b 24</code> for BeagleBone 24. Can be set multiple times to include multiple BeagleBones. The ID of the BeagleBone has to correspond to its static ip address(e.g. bb-24 has ip 192.168.1.24).
<code>-d <i>delay</i></code>	the number of measurement steps the script shall wait until to apply any kind of network or CPU load

Example

```
sudo ./startMeasurements -m -n 1000 -f test -b 24
```

This command started on the master will start a measurements with 1000 steps (1000 seconds) on master and a BeagleBone with the number 24, saving the data to the file named *test*.

A.2.2 External Measurement Script

This script changes kernel modules on the BeagleBone and sets the direction of the GPIO pin used for the external measurement, as well as restarting all the services used for the time synchronization. In normal operation (not reversed) it inserts the *pps-generator* kernel module on both master and slave and sets the GPIO direction in `/sys/class/gpio/gpio60/direction` (pin P9.12 on BeagleBone) to *out*. On the slaves it also removes the *pps-gmtimer* kernel module to reduce interference of modules. In reverse mode only the modules that previously got inserted are removed and the removed ones are inserted, but no restart of the services is initiated.

This script is meant to run on the slaves and to be used right before the start of a measurement using a logic analyzer.

Synopsis

```
./extMeasurement -rmp [-b beaglebone]
```

Options

- r reverse the operation and load the module
- m this option has to be set if the script is started on the master. It will allow it to start the same options given by other flags on other BeagleBone specified with the **-b** flag.
- p use *phc2sys* on the slaves as time synchronization instead of *chrony*.
- b *beaglebone* specify the ID of the BeagleBone to be used in this measurement, e.g **-b 24** for BeagleBone 24. Can be set multiple times to include multiple BeagleBone. The ID of the BeagleBone has to correspond to its static ip address(e.g. bb-24 has ip 192.168.1.24).

Example

```
sudo ./extMeasurement -m -r -b 24
```

This command started on the master reverses the insertions of the *pps-generator* kernel module on the master as well as the BeagleBone with ip 192.168.1.24

A.3 Data Post Processing

In this section all the scripts used for processing and displaying the data collected are explained. All the scripts are written in Python with use of the pandas, numpy, pyparsing, bokeh and matplotlib packages. The first two scripts in the first section were only used for quick display of data and the functions in the script *calculate_statistics.py* were taken and adapted in the later scripts for a more clean and better working post processing of the data.

A.3.1 Quick Displaying of Collected Data [deprecated]

The scripts *calculate_statistics.py* and *calculateHistograms.py* were used to get a quick look at the collected data. They are not used for the final plots and are deprecated, although they still work. The usage of them is briefly explained here.

calculate_statistics.py

This script takes measurement data as input and outputs a bokeh html plot website with interactive timeseries of all the collected measurements.

Synopsis

```
calculate_statistics.py [-h] [--p] [--l] [--t] folder_name
folder_name
```

Options

positional arguments:

folder_name Name of the folder which the program should calculate the statistics: 2 Arguments for nested folders: example: direct E2E-phc2sys for folder direct/E2E-phc2sys

optional arguments:

-h, -help show this help message and exit
 -p choose phc2sys file on slave instead of chrony (default:chrony)
 -l parse logic analyzer file or skip it(default:parse it)
 -t trim timescale to all be the same(default:trim it)

Example

```
python calculate_statistics.py --l --t direct E2E-UDpv4
```

This command takes all the data from the measurement in the folder *direct/E2E-UDpv4* except for the external measurement data from the logic analyzer and plots it. Every timeseries has their own timescale since it is not adjusted by the script.

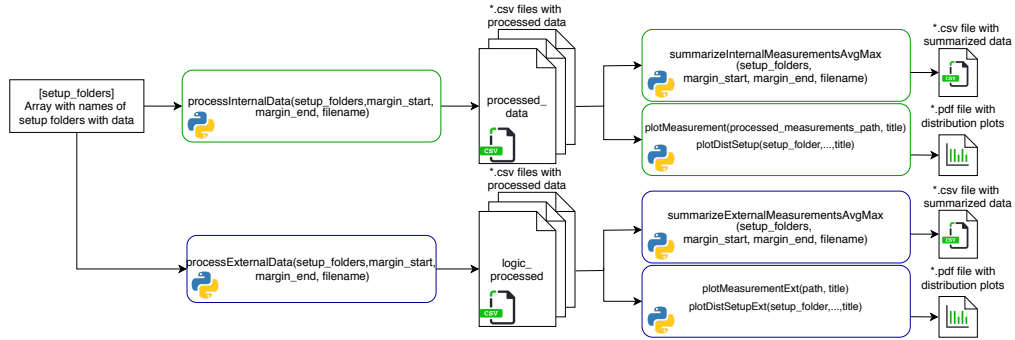


Figure A.1: The top view of the post processing workflow. Green is for internal, blue for external collected data.

calculateHistograms.py

This script takes measurement data as input and outputs a bokeh html plot document with summarized histograms containing the average and the maximum of each measurements *gmtime* data. It imports the *conver_to_gmtime_df* function of the *calculate_statistics.py*.

Synopsis

```
calculateHistograms.py [-h] [--p] [--name FILENAME] [N [N ...]]
```

Options

positional arguments:

N

Name of the folder which the program should calculate the statistics: 2 Arguments for nested folders: example: direct E2E-phc2sys for folder direct/E2E-phc2sys the first folder has to be the setup folder (e.g. "direct")

optional arguments:

-h, -help

show this help message and exit

-p

choose phc2sys file on slave instead of chrony (default:chrony)

--name FILENAME

name of output file

Example

```
python calculateHistograms.py direct BC-sCPU-50d BC-sCPU-100d
BC-CPU-50d BC-CPU-100 --name CPU-Histogram
```

This command summarizes all the data from the *gmtime* measurements of the different measurements and plots it as a grouped histogram. The name of the file will be *CPU-Histogram.html*.

A.3.2 Overview of Data Processing

Since the data, which was collected during measurements was heterogeneous and did not necessarily share the same start and endpoint of the timeseries the first

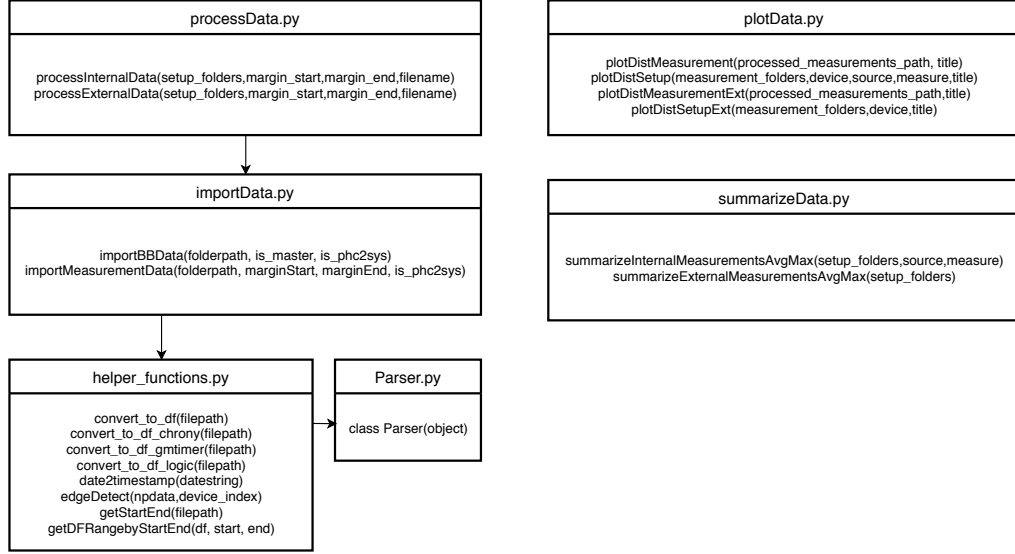


Figure A.2: The relationship of the files with their functions to one another.

step was to process this data to later be able to conveniently plot from this processed data or get the average of all measurements. This also reduced the processing effort, since the parsing of the raw data was quite intense. Another advantage of this split between the processing and the plotting is that we only have to do one post processing for many plots.

In fig. A.1 The processing of the data is depicted with the corresponding functions. On the left we see an array with the names of the folders which should be processed and contain the raw data, is processed by the functions and for each device (e.g. master or slave BeagleBone) a table with the processed data is saved in the corresponding folder. This is done for the internal and external data separately. On the right side we see the processed data is read back in by the summarizing and the plot function to produce tables with averages and maximum (like in fig. 4.1) or plots as seen in the results section of the thesis. In the next section the dependency relationship between the functions is given and every function is briefly described.

A.3.3 Overview of Files, Functions and Dependencies

In fig. A.2 The dependencies between the files and their functions is given. It is seen that because of the split between the processing and displaying the data, the corresponding functions do not directly depend on each other. The only dependency is the way the processed data is saved and read back in by the functions has to be consistent. In the following the functions from each file are briefly explained.

helper_functions.py

This file contains small helper functions and the basic functions for the import of raw data.

convert_to_df(filepath)

Convert the raw data from the *ptp4l.log* or the *phc2sys.log* file to a pandas DataFrame.

Parameters: **filepath:** *string*

The exact filepath to the log file

Returns: **DataFrame**

Dataframe with timestamps as index and columns *offset* and *delay* or *rms,max,delay,delay_dev* depending on the raw data

convert_to_df_chrony(filepath)

Convert the raw data from the *chrony.log* file to a pandas DataFrame.

Parameters: **filepath:** *string*

The exact filepath to the *chrony.log* file

Returns: **DataFrame**

Dataframe with timestamps as index and columns *offset* and *offset_std_dev*

convert_to_df_gmtimer(filepath)

Convert the raw data from the *gmtimer.csv* file to a pandas DataFrame.

Parameters: **filepath:** *string*

The exact filepath to the *gmtimer.csv* file

Returns: **DataFrame**

Dataframe with timestamps as index and column *offset*

convert_to_df_logic(filepath)

Convert the raw data from the *logic.csv* file to a pandas DataFrame using the *edgeDetect* function

Parameters: **filepath:** *string*

The exact filepath to the *gmtimer.csv* file

Returns: **DataFrame**

Dataframe with no explicit index and the columns *master,slave24, slave25, slave26* which contain the calculated time offset for each point in time

date2timestamp(datestring)

Convert a date in the form "%Y-%m-%d %H:%M:%S" to a timestamp

Parameters: **datestring:** *string*
The string to be parsed and converted

Returns: **Timestamp:** *Int*
Timestamps (seconds since origin)

edgeDetect(npdata,device_index)

Detect a rising edge in the raw logic analyzer data and calculate the offset.

Parameters: **npdata :** *numpy array*
The numpy array with the raw data from the logic analyzer

device_index : *Int*
The index of the column of the device of which we want to calculate the offset of.

Returns: **Panda Series:** *pd.Series*
Panda series with all the offsets calculated in nanoseconds

getStartEnd(filepath)

Get the start and end time of the measurement from the *time.txt* file

Parameters: **filepath :** *string*
The path to the folder of the *time.txt* file

Returns: **(start,end)**
The start and end time as specified in the *time.txt* file, as a tuple of timestamps

getDFRangebyStartEnd(df, start, end)

Get a range of a DataFrame using start and endtime given as integers.

Parameters: **df :** *pd.DataFrame*
The DataFrame which we like to project from

start:*Int*
The timestamp of the starting time

end: *Int*
The timestamp of the end time

Returns: **DataFrame**
The range from the Dataframe which lies in the timerange of start and endtime given

Parser.py

Parser Class

A Parser object used to parse systemlog files. It has an initialization function and a function *parse(self,line)*. The code was adapted from github³

³<https://gist.github.com/leandrosilva/3651640>

Parser.parse(self,line)

Parse a line of the syslog file.

Parameters: **line** : *string*

A single line from the syslog as a string

Returns: **Dict**

Returns the different parts of the systemlog as a dictionary with keys *timestamp*, *hostname*, *appname* and *message*

importData.py

In this file the two functions used to import the raw data are located.

importBBData(folderpath, is_master, is_phc2sys)

Imports the raw data from a single device/BeagleBone and outputs a single table in the form of a DataFrame.

Parameters: **folderpath** : *string*

The path to the folder containing the raw data.

is_master:*boolean*

If the folder and measurement belong to a master device then it should be set to True, False else.

is_phc2sys:*boolean*

If the measurement was done using *phc2sys* instead of *chrony* this Boolean should be True, False otherwise.

Returns: **Dataframe**

Returns a DataFrame containing all measured values in a corresponding column with the timestamps as index. This DataFrame has multicolumns for each measurement and the corresponding measure.

importMeasurementData(folderpath, marginStart, marginEnd, is_phc2sys)

Imports the raw data from a whole measurement and outputs a single table in the form of a DataFrame.

Parameters: **folderpath** : *string*
The path to the folder of the measurement containing the folders of the devices with the raw data from the measurements in it.
marginStart:*Int*
Number of values to be dropped at the beginning of the table
marginEnd:*Int*
Number of values to be dropped at the end of the table
is_phc2sys:*boolean*
If the measurement was done using *phc2sys* instead of *chrony* this Boolean should be True, False otherwise.

Returns: **Dataframe**
Returns a DataFrame containing all measured values from all the devices in a corresponding column with the timestamps as index. This DataFrame has multi-columns for each device, measurement and the corresponding measure.

processData.py

This file contains the toplevel functions to process the raw data to processed data.

processInternalData(setup_folders,margin_start,margin_end,filename)

Processes all the raw data from the internal measurements, drops the given number of rows at the beginning and end of the table and writes it to a file.

Parameters: **setup_folders** : *Array of strings*
An array of the paths to the folders of the measurements containing the folders of the devices with the raw data from the measurements in it.
margin_start:*Int*
Number of values to be dropped at the beginning of the table
margin_end:*Int*
Number of values to be dropped at the end of the table
filename:*string*
Name of the output file

processExternalData(setup_folders,margin_start,margin_end,filename)

Processes all the raw data from the external measurements, drops the given number of rows at the beginning and end of the table and writes it to a file.

Parameters: **setup_folders** : *Array of strings*
 An array of the paths to the folders of the measurements containing the folders of the devices with the raw data from the measurements in it.
margin_start: *Int*
 Number of values to be dropped at the beginning of the table
margin_end: *Int*
 Number of values to be dropped at the end of the table
filename: *string*
 Name of the output file

summarizeMeasurements.py

This file includes all functions which are used to summarize measurements.

summarizeInternalMeasurementsAvgMax(setup_folders,source,measure)

This function outputs a DataFrame with the average and maximum of the column specified with *source* and *measure* over all the setups given via the *setup_folders* array. The sources are the internal measurements.

Parameters: **setup_folders** : *Array of strings*
 An array of the paths to the folders of the measurements containing the folders of the devices with the raw data from the measurements in it.
source: *string*
 The name of the column which corresponds to the source of the measurement data, e.g. *gmtimer*.
measure: *string*
 The name of the column from the measure of the source, e.g. *Offset*

Returns: **Dataframe**
 Returns a DataFrame containing all the averages and maximums from all devices and all setups given

summarizeExternalMeasurementsAvgMax(setup_folders)

This function outputs a file with the average and maximum of the external measurement data given for all the setups given via the *setup_folders* array.

Parameters: **setup_folders** : *Array of strings*
 An array of the paths to the folders of the measurements containing the folders of the devices with the raw data from the measurements in it.

Returns: **Dataframe**
 Returns a DataFrame containing all the averages and maximums from all devices and all setups given

plotData.py

This file contains all functions used for plotting.

plotDistMeasurement(processed_measurements_path, title)

Writes to a *matplotlib* object the distribution of the *gmtimer,offset* column for all devices of a measurement.

Parameters: **processed_measurements_path** : *string*
 The path to the processed data file.
title : *string*
 Title of the plot.

plotDistSetup(measurement_folders,device,source,measure,title)

Writes to a *matplotlib* object the distribution of the specified column for the specified device of all the measurements in a setup.

Parameters: **processed_measurements_path** : *string*
 The path to the processed data file.
device:*string*
 The name of the column which corresponds to the name of the measurement device, e.g. *slave24*.
source:*string*
 The name of the column which corresponds to the source of the measurement data, e.g. *gmtimer*.
measure:*string*
 The name of the column from the measure of the source, e.g. *Offset*
title : *string*
 Title of the plot.

plotDistMeasurementExt(processed_measurements_path,title)

Writes to a *matplotlib* object the distribution of the column for all devices of a measurement.

Parameters: **processed_measurements_path** : *string*
 The path to the processed data file.
title : *string*
 Title of the plot.

plotDistSetupExt(measurement_folders,device,title)

Writes to a *matplotlib* object the distribution of the specified column for the specified device of all the measurements in a setup.

Parameters: **processed_measurements_path** : *string*

The path to the processed data file.

device:*string*

The name of the column which corresponds to the name of the measurement device, e.g. *slave24*.

title : *string*

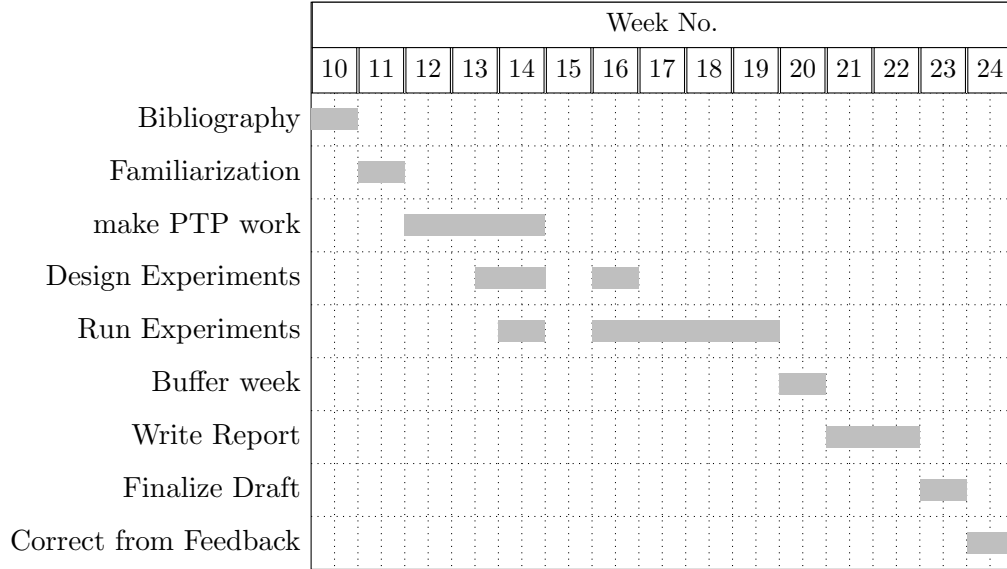
Title of the plot.

A.4 Known Issues

- PHC changes form ptp0 to ptp1 and back
maybe caused by already running instance of *ptp4l*
- P2P does not work
might be hardware issue ⁴
- Failed clock creation: Rarely, *ptp4l* failed on startup because it could not create a clock
maybe caused by an already running instance of *ptp4l*

⁴<https://e2e.ti.com/support/processors/f/791/t/650432?Linux-TMDXIDK5728-PTP-issue>

A.5 Timetable



Bibliography

Collect Papers for similar Project and necessary Standard Documentation and read them.

Familiarization

Get familiar with the beaglebones and chrony.

make PTP work

Make a first Version with two beaglebones directly connected work and try to measure it.

Design Experiments

Design different experiments and setup a measurement framework.

Run Experiments

Execute the different experiments and gather data.

A.6 Original Project Assignment



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



Semester Thesis at the
Department of Information Technology and
Electrical Engineering

for

Julian Huwyler

PTP Time Synchronization for FlockLab 2

Advisors: Roman Trüb
Reto Da Forno

Professor: Prof. Dr. Lothar Thiele

Handout Date:	20.01.2020
Official Start Date:	06.03.2020
Due Date:	12.06.2020

1 Project Description

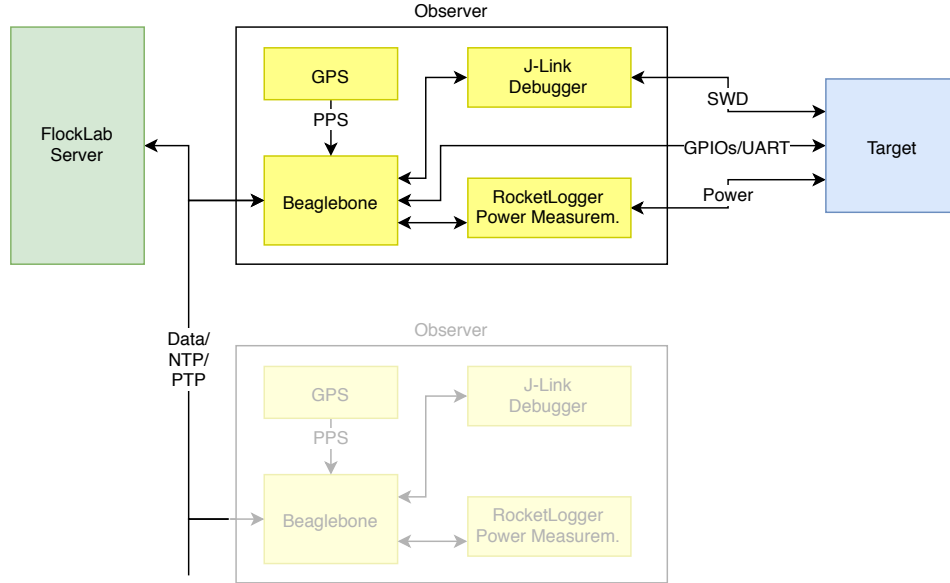


Figure 1: FlockLab 2 Architecture.

Since 2012, the Computer Engineering Group (TEC) operates the FlockLab testbed [4] for developing and evaluating wireless sensor network protocols. A testbed helps to reduce the effort of repeatedly deploying test networks when developing protocols for wireless sensor networks. Furthermore, such a testbed improves the reproducibility of experiments and allows to share infrastructure.

Currently, we are in the process of extending the existing short baseline distances in FlockLab by adding additional nodes on rooftop locations and with significantly larger spacing. Since the existing implementation of FlockLab is based on hardware components which are no longer in production we developed the next generation of the FlockLab, FlockLab 2. The new architecture is depicted in Figure 1. Among other improvements, FlockLab 2 incorporates a Linux platform with more performance (Beaglebone Green), more precise power tracing based on the RocketLogger [5], as well as state-of-the-art debugging support. In addition, it extends the possibilities for time synchronization of the observers.

Since the testbed is a system of distributed observers which are used to develop and debug a network wide wireless protocol, the time synchronization of the obtained measurement data is very important. We target a time synchronization accuracy of approximately 1 microsecond. The new platform features a GNSS receiver which provides accurate time synchronization at locations with sufficient GNSS reception. For locations without sufficient GNSS reception, the *Precision Time Protocol* (PTP) [2] over Ethernet can be used to synchronize the time of the observer. PTP has the potential to provide sub-microsecond accuracy.

2 Project Goals

The goals of this project are:

- Getting PTP time synchronization running on Beaglebone single board computers with the Debian operating system.
- Characterization of the achievable time synchronization accuracy and quality.

3 Project Tasks

- Formulate a time schedule and milestones for the project. Discuss and approve this time schedule with your supervisors.
- Search and read related work (e.g. [3]) and familiarize yourself with the PTP standard[2].
- Familiarize yourself with the Beaglebone single-board computer, the Debian Linux system, and the FlockLab 2 hardware setup.
- Work out a working PTP time synchronization setup with 2 Beaglebone Green single board computers running Debian 9 (or later). Investigate different configuration options (e.g. with/without hardware time stamping support, etc.). The idea is to use the provided Linux kernel (i.e. not using a custom compiled kernel) and to use the `chrony`[1] time service implementation.
- Design (with the help of the supervisors) and execute experiments to characterize the time synchronization accuracy achieved with PTP on the Beaglebone platform. Consider different setups with potentially different time synchronization properties:
 - Directly connect two Beaglebones via an Ethernet cable.
 - Connect two Beaglebones using a hub, a switch, a router.
 - Synchronize the time of more than 2 Beaglebones using PTP.
- Document your project with a written report. As a guideline, your documentation should be as thorough to allow a follow-up project to build upon your work, understand your design decisions taken as well as recreate the experimental results.

4 Project Organization

Deliverables

- Time schedule (2 weeks after start date)
- Initial presentation (3 min)
- Final presentation (15 min)

- Weekly report, which includes: current progress, problems encountered and next steps.
- Code of implementation including documentation
- Final report, which includes: introduction, analysis of related work, documentation of decisions, evaluation, description and HowTo guide of the developed software.

Offers

- The supervisors offer the student the opportunity to do a rehearsal of the initial and the final presentation. The supervisors offer to give feedback how to improve the presentations.
- The supervisors offer to proof-read a draft of the final report. The draft is not required to be complete. The draft should be handed in no later than 1 week before the due date of the thesis.

General Requirements

- The project progress shall be regularly monitored using the time schedule. Unforeseen problems may require adjustments to the planned schedule. Discuss such issues openly and timely with your supervisors.
- Use the work environment and IT infrastructure provided with care. The general rules of ETH Zurich (BOT) apply. In case of problems, contact your supervisor.
- Code versioning is **mandatory** throughout the thesis and the student is responsible for regularly pushing her/his contributions to the repository.

Weekly Meeting

- At the beginning of the thesis, a time slot for the weekly meeting will be agreed on. The weekly meeting is used to discuss the project's progress based on a schedule defined at the beginning of the project.
- The weekly report should be provided at latest at 23:59 on the day before the weekly meeting.

Initial Presentation

Prepare the initial presentation which should include:

- Short introduction (e.g. name, origin, previous studies, current study status, why you are interested in this topic)
- Short description of the project (What do we do, why do we do it and why is it hard)

- Project goals (What do we want to achieve)
- Intended way to reach goals (How do we want to achieve it)

The presentation must not exceed 3 minutes (approx. 3 slides with content). A date for the presentation will be assigned during the project.

Final Presentation

Prepare the final presentation which needs to include:

- Short project description and project goals
- Detailed presentation of the work done
- Detailed presentation of the results
- Conclusion and outlook on possible future work

The presentation must not exceed 15 minutes. A date for the presentation will be assigned during the project.

Handing In

- Hand in a single PDF file of your project report via email. In addition, hand in the signed declaration of originality on paper. A hard-copy of the report is not required.
- Clean up your digital data in a clear and documented structure using the provided GitLab repository. In the end, all digital data should be contained in the student's GitLab repository for the thesis. This includes: developed software, measurements, presentations, final report, etc. An exception are large amounts of measurement data which is stored separately (ask your supervisors!).

References

- [1] chrony. <https://chrony.tuxfamily.org/v>.
- [2] IEEE 1588-2008 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, 2008.
- [3] K. Geissdoerfer, M. Chwalisz, and M. Zimmerling. Shepherd: a portable testbed for the batteryless iot. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, pages 83–95, 2019.
- [4] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks*, IPSN '13, pages 153–166, New York, NY, USA, 2013. ACM.

- [5] L. Sigrist, A. Gomez, R. Lim, S. Lippuner, M. Leubin, and L. Thiele. Measurement and validation of energy harvesting iot devices. In *Proceedings of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE 2017)*, Lausanne, Switzerland, Mar 2017.

Bibliography

- [1] Cisco annual internet report (2018–2023) white paper. URL <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>.
- [2] Ieee standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pages 1–300, 2008.
- [3] Kai Geissdoerfer, Mikołaj Chwalisz, and Marco Zimmerling. Shepherd: A portable testbed for the batteryless iot. 11 2019. ISBN 978-1-4503-6950-3. doi: 10.1145/3356250.3360042.
- [4] Antonio Libri, Andrea Bartolini, and Luca Benini. Dwarf in a giant: Enabling scalable, high-resolution hpc energy monitoring for real-time profiling and analytics. 06 2018.
- [5] Antonio Libri, Andrea Bartolini, Daniele Cesarini, and Luca Benini. Evaluation of ntp/ptp fine-grain synchronization performance in hpc clusters. In *Proceedings of the 2nd Workshop on Autotuning and ADaptivity AppRoaches for Energy Efficient HPC Systems*, ANDARE ’18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450365918. doi: 10.1145/3295816.3295819. URL <https://doi.org/10.1145/3295816.3295819>.
- [6] Antonio Libri, Andrea Bartolini, Michele Magno, and Luca Benini. Evaluation of synchronization protocols for fine-grain hpc sensor data time-stamping and collection. In *2016 International Conference on High Performance Computing & Simulation (HPCS)*, pages 818–825. IEEE, 2016.
- [7] Robert Manzke Mudassar Ahmed. Implementation and performance analysis of precision time protocol on linux based system-on-chip platform. 9 2018.
- [8] J Serrano et al. The white rabbit project, proceedings of the 2nd international beam instrumentation conference, 2013.
- [9] Roman Lim, Federico Ferrari, Marco Zimmerling, Christoph Walser, Philipp Sommer, and Jan Beutel. Flocklab: a testbed for distributed, synchronized tracing and profiling of wireless embedded systems. pages 153–166, 04 2013. doi: 10.1145/2461381.2461402.

- [10] Flocklab 2. URL <https://flocklab.ethz.ch>.
- [11] Rocketlogger. URL <https://rocketlogger.ethz.ch/>.
- [12] Beaglebone pru. URL <https://beagleboard.org/pru>.
- [13] Beaglebone black hardware counter capture driver(pps-gmtimer). URL <https://github.com/kugelbit/pps-gmtimer>.
- [14] gpsd — a gps service daemon. URL <https://gpsd.gitlab.io/gpsd/index.html>.
- [15] chrony. URL <https://chrony.tuxfamily.org/>.
- [16] Saleae logic 8, logic analyzer. URL <https://www.saleae.com>.
- [17] stress-ng. URL <https://manpages.debian.org/testing/stress-ng/stress-ng.1.en.html>.
- [18] iperf. URL <https://iperf.fr/>.
- [19] Planet gsd-805 gigabit ethernet switch, . URL <https://www.planet.com.tw/en/product/gsd-805>.
- [20] Ubiquiti unifi switch 8, . URL <https://www.ui.com/unifi-switching/unifi-switch-8/>.