M Ű E G Y E T E M 1 7 8 2

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Bálint Ferencz

Hardware Assisted IEEE 1588 Clock Synchronization Under Linux

Master's Thesis

Supervisors

Tamás Kovácsházy Dr.
Dept. of Measurement and Inf. Systems

Zoltán Fodor
Intel Hungary

Budapest, 2013

Contents

Abstract

This thesis presents a solution for reliable and cost-effective implementation of a high accuracy time synchronization system in Local Area Networks via the IEEE 1588 protocol. Nowadays the necessity of sub-microsecond time synchronization is increasing in the fields of distributed systems and multimedia systems. By the usage of the general purpose communication protocols such as Ethernet for synchronization, the cost and complexity can be lower compared to ordinary systems (e.g. via GPS and IRIG synced nodes).

There are several current applications of high accuracy in–band clock synchronization such as *power substation automation*, *high frequency trader financial systems*, *telecommunications*, etc. This thesis shows a reliable solution for Linux based systems to attain this goal. The Ethernet based *multimedia systems* provide a less complex and cost effective alternative to the multimedia specific connections based systems by eliminating the proprietary and single-function connections (VGA, DVI, HDMI) with the help of packet based clock synchronization.

The document systematizes the earlier works in this field, and presents an IEEE 1588–hardware time stamping based solution for the high accuracy i.e. sub-microsecond synchronization. A high accuracy clock output (1PPS) is designed and presented in this work. This output can be used for the distribution of the correct time to other devices, which can't use the PTP protocol stack, and it's also used for the validation of the measurements.

Every clock sync system requires a high precision time source for the most accurate synchronization process. In the measurements, the demo system contains a high precision GPS based PTP master clock hardware. The thesis also shows the design and implementation issues of a substituting equipment.

The performance of the network routing/switching equipment is critical for the decent accuracy. The tests of the system are performed with PTP & non PTP compliant switching devices with variable loads to cover the largest real world application cases.

List of terminology

accuracy	The maximal difference between an arbitrary node & the reference time
BC	Boundary Clock
COTS	Commercial off the shelf
DAC	Digital–analog converter
discipline	Steering the phase/frequency of a clock to a reference
DUT	device under test
FOSS	Free and open-source software
GPS	Global Positioning System
ioctl	A Linux kernel command/interface to program any kind of hardware
ISR	Interrupt Service Routine
OCXO	Oven Controlled Crystal Oscillator
PHC	PTP hardware clock
PLL	Phase locked loop
ppb	Particle per billion
ppm	Particle per million (onehundredth of a percent)
PPS	Pulse per second
precision	The smallest quantum of time representable in the system
PTP	Precision Time Protocol, name alternative to IEEE 1588
SDP	Software Definable Pin

SHM	Shared Memory
TAI	International Atomic Time
TC	Transparent Clock
UTC	Coordinated Universal Time
VM	virtual machine
wrt.	with respect to

1 | Introduction

"A man with a watch knows what time it is. A man with two watches is never sure."

— Segal's Law

1.1 Motivation

The emerging importance of maintaining a stable, accurate timebase in distributed systems requires a supporting infrastructure to distribute the common wall clock time to the nodes. The usage of the existing communication network makes these systems cheaper and less complex, therefore the expected reliability is higher and the administration overhead can be smaller.

Several solutions appeared in the past to satisfy the required functionality — the venerable NTP provides a widespread, reliable solution of the synchronization over Internet, but it doesn't meet the most rigorous accuracy requirements. Distribution of sub–microsecond accuracy timing information with out–of–band communication is available long since; the U.S. originated GPS is among the most widespread solution to provide reference time, but its indoor usage is limited. Vendors are selling proprietary solutions for solving the in–band time synchronization needs, but currently the open-source alternatives are covering only parts of the needed functionality.

The main goal of this thesis to introduce the main concepts behind the high accuracy clock synchronization in non-realtime systems, and to document the development & integration of the necessary open-source components to build a high precision time source based on PC architecture. The presented solution is based on GPS and the IEEE 1588 protocol standard; it runs on Linux operating system.

I was introduced to concept of packet based time synchronization in 2010 when I chose my first project laboratory topic. Back then there were several commercial solutions to assemble PTP based synchronization networks. There was an open–source application called *PTPd* [1] which implemented the first IEEE 1588 standard's master and slave clock roles. My first task was to get comfortable with the basics of the PTP and get the PTPd daemon working with hardware time stamping capabilities — the main goal was to discipline the kernel clock as good as possible.

In the following years a common API for controlling the NIC's onboard clocks (*PHC API* [2]) were introduced along with a new *linuxptp* [3] project. From that on a maintained IEEE 1588:2008 compliant FOSS daemon were also available to use. The implementation effort of high–quality and cheap computer based clocks becomes easier and easier, and I was involved in implementation and measurement of PTP based clock synchronization systems. This thesis is a natural evolution all of my previous findings about this subject.

1.2 Analysis of problem statement

The proposition of the thesis was to examine the current solutions of the PTP based clock synchronization networks, and to propose a solution to attain sub-µs accuracy time synchronization with the usage of freely available software tools and Intel network interface cards. In order to properly document the proposed tasks, I will organize the this document in the following manner:

1. The technologies behind the topic (IEEE 1588, Linux time handling, GPS) will be introduced to the reader.
2. The general system hierarchy will be discussed with the identification of the roles of the major building blocks — with the comparison of possible advantages/drawbacks.
3. The implementation details of my work will be presented.
4. The measurement results will be discussed before the conclusion.
5. In the conclusion part I will introduce application scenarios of the product and summarize most important findings in this document and the future ways of research.

I've focused on the maximum attainable accuracy with the usage of COTS components in PTP networks, the analysis of security problems are outside of the scope of this paper.

1.3 Standards and technologies involved

1.3.1 Brief summary about the time metrics

The meausurement of the flow of time dates back to the ancient civilizations who tried to formalize cyclic events of the days and years. The most apparent way of the time measurement was the counting of the sunrises and sunsets, but even the human perception spots the varying length of time between these events, and that is the most important drawback of empiric/astronomical definition of the unit of time. As science and technology advanced, better and better models and theories emerged about our world, which implied more precise definition of the time itself. The days derived from the observation of the sky were replaced by an exact physical phenomenon as the fundamental metric; nowadays the basic unit of time is the *second*, which is defined as 9,192,631,770 cycles of radiation corresponding to the transition between two hyperfine levels of the ground state of cesium 133 [4].

The *International Atomic Time* (TAI) is a statistically derived unit based on the counting of the previously defined seconds with a large number of atomic clocks. The *Universal Time* (UT0, UT1) is counted from 0 hours at midnight, with unit of duration the mean solar day, defined to be as uniform as possible despite variations in the rotation of the Earth. The UT0 is the observed rotational time of an extraterrestrial point viewed from a fixed location on Earth and the UT1 is the corrected measurement of the former with the irregularities of the Earth's motion. *Coordinated Universal Time* (UTC) differs from TAI by an integral number of seconds. UTC is kept within 0.9 seconds of UT1 by the introduction of one-second steps to UTC, the *"leap second"*. To date these steps have always been positive. UTC has replaced Greenwich Mean Time (GMT) as an international standard [4].

1.3.2 Overview of IEEE 1588

The IEEE 1588 standard describes a protocol which synchronizes computer clocks via Multicast capable networks [5]. The protocol became standard in 2002 but it was revised in 2008; the resulting update is not backwards compatible with the original version. With proper hardware support the sub-microsecond synchronization accuracy is attainable in the scope of local area networks. The recommended timescale of PTP is the same as TAI, which simplifies the implementation of the measurement systems. This timescale may be changed to an arbitrary one (such as UTC), but several subprofiles forbid the usage of other ones. The standard offers an automatic way for the configuration of the synchronization network, based on the nodes preprogrammed capabilities. A synchronization network consists one or more of the following entities:

- Master clock
- Slave clock
- Boundary clock (optional)
- Transparent clock (optional)

Every clock synchronization network has at least one *master*, and one *slave* clock. A synchronization network can be divided into several smaller separate domains where the synchronizing slaves obtain their wall time from discrete masters. Primary master of the whole synchronization network is called *grandmaster*, the slaves and masters are also called *ordinary clocks*. The synchronization algorithm supports two main modes of function, the *one step* and *two step* synchronization. The masters of the network send only *sync* messages when they operate in one step synchronization mode, but they send a follow-up messages when not.

With the usage of the *follow-up* messages, the master can send the precise sending time of the sync packets to the slaves if it's unable to inject this information in the sync packets on the fly. By sending follow-up messages the accuracy is improved with a negligible increase of the network traffic (Fig. 1.1.). The slave nodes of the network capture the ingress times of the sync packets. Now the offset from the master can be expressed as:

$$\Delta t = T_{s1} - T_{m1} = 49\ s \tag{1.1}$$

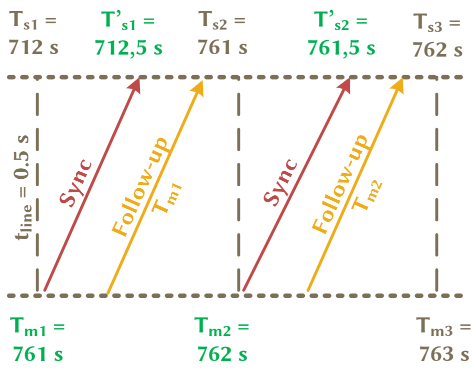


Figure 1.1: The offset correction algorithm

The applied offset corrections have constant error because the signalling delay is currently disregarded. Estimation of the path delay is requested by the slaves via *delay* *request* messages, the slave records the sending time of the packet, and the receive time of the master's *delay response* (Fig. 1.2.). The master sends back the reception time of the delay request package to the slave. It is assumed that the link delay is symmetrical, and it's variance is low — this assumption is generally acceptable in local area networks, but in high accuracy systems the asymmetries of the system should be eliminated or compensated. The estimator of the delay is the mean of the master-to-slave and slave-to-master propagation times. In real world applications both the offset and delay calculations include delicate filtering algorithms to cancel out the noise caused by the jitter of time stamping, or the network delay.

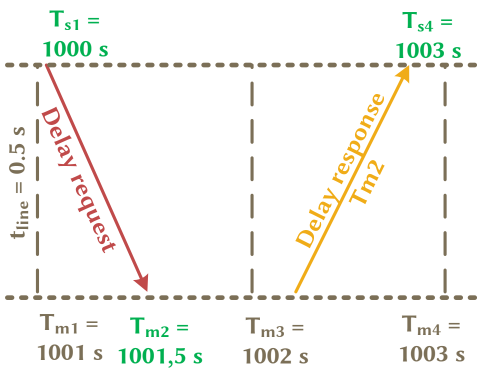


Figure 1.2: The delay estimation algorithm

The autoconfiguration of the system is done with the *Best Master Clock* (BMC) algorithm. Every PTP port maintains a dataset about its capabilites, the most important fields are (in relative order of precedence):

- `clockClass`,
- `clockAccuracy`,
- `offsetScaledLogVariance`
- `clockID`.

The `ClockID` is the tie–breaker in the selection, its generated from the (presumably) unique MAC address of the adapter in most cases. If we measure the Allan deviation of the clock we can provide the `offsetScaledLogVariance` field, which describes the quantitative performance of the clock ([5] pp. 58). The `clockAccuracy` represents an interval of maximum error compared to the reference time when the PTP port works as a grandmaster. The `clockClass` field represents the traceability of the time provided the PTP clock when it works as a grandmaster.

Clock synchronization regions could be linked together by *boundary clocks*. These devices act as PTP slave on maximum one of their PTP logical ports, and as masters on every other (Fig. 1.3.). In some cases the ports are in passive — non-functional — state to prevent unwanted synchronization loops in the network. Their local time — which is shared among the logical ports — was corrected by the slave port of the device, and it serves as timesource for the master ports. PTP messages of the different regions handled separately, the logical ports of the domains do not know about the syncing nodes in other regions (it's different for PTP management messages which are relayed through the ports of the boundaries). The segmentation helps to lower the traffic caused jitter in individual segments compared to a larger network, and also helps to integrate different communication media into a single synchronization entity.

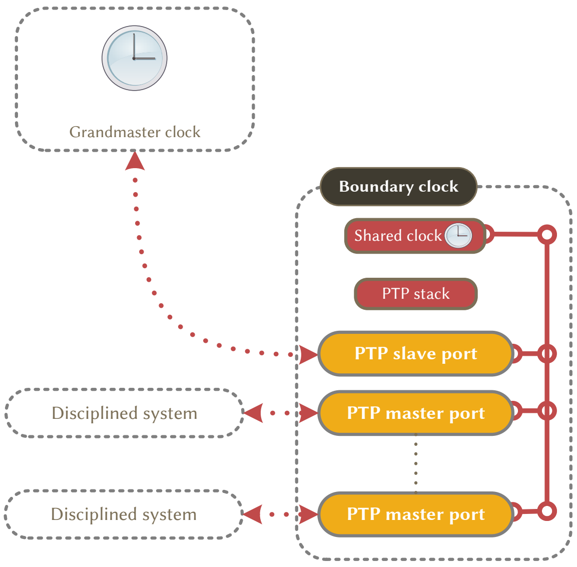


Figure 1.3: Block diagram of a boundary clock

The revised standard (IEEE 1588:2008) introduced the concept of *Transparent Clock* (TC) (see Figure 1.4.); it's used to mitigate the packet jitter caused by the non–determinstic residence time in the active elements. These devices can measure of the time of residence between their ingress and egress ports (T_1 - T_2 and T_1 - T_3), and update the follow-up packets with this information. Also the transparent clocks can maintain the correct order of the transmitted sync messages and the corresponding follow-up messages [6]. To ensure the correct measurement of the residence time, the local clock of the TCs should be synthonized (frequency corrected) to the master of their respective segment. There are two main types of Transparent Clocks, the End to End devices update the `CorrectionField` value of the necessary packets (`sync`, `delay_req`, `follow_up`) with the residence time, and each slave computes the path delay individually with respect to the grandmaster. With the usage of Peer to Peer Transparent Clocks, the slave ordinary clocks can measure their direct path delays to their nearest TC. It means that every active node in the system computes the link delay to the nearest upstream port only, and the slave system doesn't compute the link delay from the grandmaster, but uses the cumulated `CorrectionField` to estimate the link delay on the path.

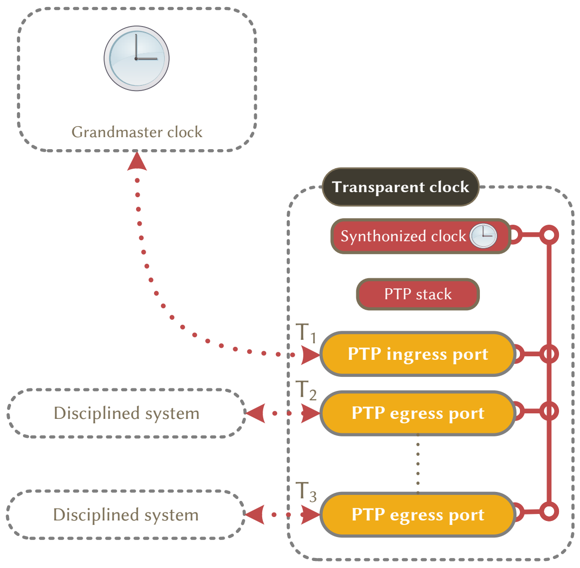


Figure 1.4: Block diagram of a transparent clock

1.3.3 Overview of the timekeeping in Linux

Like every operating system, Linux also has some sort of abstraction of time [7][8]. The operating system is based on event–based principles, but the abstraction of the *real time* is also present.

The Linux kernel time is represented by the *jiffies* counter on every architecture. Based on the kernel settings the frequency of a jiffy — in terms of the real time — varies in the range of milliseconds, therefore it cannot be used alone to measure events with high precision. The incrementing jiffies represent the number of the timer interrupts of the system since the last boot. For example on x86 architecture, an integrated timestamp counter of the CPU used as a source of a more granular event scale. The jiffies counter measure the elapsed cycles from the last boot, therefore the problem of the epoch handling should be also handled. It's used to measure the distance between discrete events in the kernel, or provide time base for various system calls (e.g. `select()`) but it's unsuitable to directly derive a precise representation of the real–world time from it.

The kernel specifies interfaces to estimate and query the *real time* of the call. In the Unix world, time is measured on the UTC scale with the epoch of 1st of January 1970, this is the so called *Unix time*. The UTC time scale means that the problem of leap seconds should be handled — there are 86400 Unix time seconds per day no matter how the timescale varies. There are several solutions to represent this kind of discontinuity in the flow of time, one can add seconds to the time scale (after minute's 60th second the 61st is inserted), repeat a second (like NTP does) or just increase the length of the seconds (time–smearing used by e.g. Google [9]). It is also problematic that the jiffies counter granularity if not enough to represent sub-millisecond time duration, but since Linux 2.6.21 the introduction of *high resolution timers* (HRTs) solves the problem of such small quantities with a common API.

Message based time synchronization needs an interface to query the transmit and receive time of the synchronization data in a real-world timescale. The Linux kernel socket interface supports the receive timestamping since the 2.4 kernel series (`SO_TIMESTAMP` or `SO_TIMESTAMPNS` – µs vs. ns timescale). The transmit timestamping is supported through looping the sent packet back to the receive queue. In 2008 the `SO_TIMESTAMPING` interface were introduced to the Linux kernel, to enable the support of the hardware timestamping. In this case the send timestamps are looped back via the error queue of the socket. The comparison of methods of the timestamping alternatives will be discussed later.

1.3.4 GPS in high precision clock synchronization

The development of the Global Positioning System was driven by the military needs of the United States during the Cold War. The measurement of position originates in the high precision measurement of the radio signal propagation times from the satellites to the receivers. To understand the connection between the time and position measurement, I will present the main ideas behind the system [10]. The layout of the system consists of three main parts — the Space Segment, the Control Segment and the User Segment.

The *Space Segment* contains all of the GPS satellites, which are orbiting around the Earth in medium Earth orbit (approx. 20.000 km away, see Fig. 1.5.). Each satellite has atomic clocks onboard to produce their own 10.23 MHz fundamental frequency. The satellites transmit their data at two distinct microwave frequencies (L1 band at 1575.42 MHz, L2 band at 1227.60 MHz). Both frequencies are created by multiplying the fundamental frequency with integer ratios (L1 – 154, L2 – 120). By the usage of the two distinct frequencies, the ground equipment can measure the deviation of the propagation times, thus they reject the measurement noise caused by the propagation in the ionosphere (it's available only for selected (military) users). Three types of code transmitted at these frequencies — the Navigation Message, the Course Aquisition (C/A) data, and the P ("precise") code. The generated data were not only transposed to those carriers, but multiplied with pseudorandom sequences (*Gold code*), whose statistical properties simplifies the reconstruction of the signal on the ground.

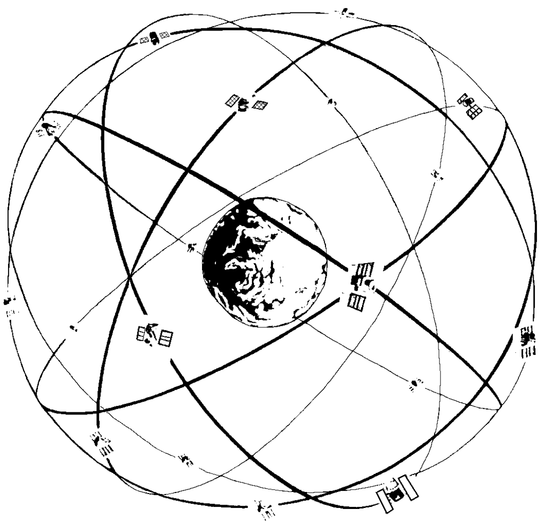


Figure 1.5: The Space Segment of the GPS [11]

The cyclic C/A data is transmitted at 1.023 Mbps (cca. every 1 ms), which contains the time when the signal was transmitted, encoded into a pseudo-random code on the L1 carrier frequency. The P code is similar to the C/A code, but its resolution is ten times higher, and the carried data is encrypted, therefore its usage is restricted to selected (military) users as of yet. The P code is delivered on both frequencies. The encryption guarantees, that the GPS aided systems — such as guided weapon systems — are protected against spoofing attacks. Unfortunately, there exist effective tools against the integrity of the GPS network based on an irreversible decalibration of the clocks [12]. The Navigation Message contains the current position of the satellite, the crude orbital data of all satellites (almanac), and the clock correction informations of the satellite. It is transmitted on the L1 channel at low speeds (50 bps) — the main reason behind the slow cold–boot (tens of minutes) of a GPS receiver, which has outdated almanac ([13] pp. 189).

The *Control Segment* plays a crucial role in the maintaining the high accuracy position data. The segment periodically calculates the current position of the satellites and sends the updated data back to them. The Earth–satellite communication uses a third carrier frequency and it is not used directly in the time measurement. With the periodically updated position data, the satellites can estimate their current position with good precision. The segment also issues commands to the satellites, such as changing orbital parameters or selecting another onboard clock in case of failure.

The *User Segment* contains the software and hardware systems of the observers. Observers of the navigation messages can reconstruct their position by measuring the clock difference of their equipment from the satellites'. At reception the reciever correlates the incoming signals to the locally generated reference signal. When the correct time displacement is found, the output of the correlator is near +1 and 0 otherwise. The receiver can calculate the distance (pseudorange) from the satellites by using the clock differences; its position is in the intersection of the calculated spheres. In simple models, the clock difference is encumbered by the errors of the satellite's clock, and the deficiencies of the local clocks. Generally a clock reading can be split into two components:

1. the actual state of the clock at time of reading,
2. the clock bias, eg. the readout incertainities, propagation delay.

If we disregard the clock biases, the position could be approximated within reasonable bounds, but in precision clock synchronization systems those informations are invaluable to reconstruct the satellites' correct clock phase and frequency information. The pseudorange calculation is sensitive to the error of the reconstructed clocks, thus delicate algorithms were developed to cancel the incertainities in the time synchronization. The GPS system uses the *Composite Clock* (CC) as its timescale, which is steered to the UTC time globally [14]. The CC is calculated from the average of the ground monitoring stations and the satellites onboard clock's. The representation of time in the GPS is counted in weeks and seconds within the week.

2 | Proposed system hierarchy

In this section the main components of the synchronization system will be discussed in depth (Fig. 2.1.). The design of the network should suit the following requirements:

- At least 1 µs accuracy (maximum difference between an arbitrary node and the grandmaster)
- Configuration free design at run-time
- Compliance with the IEEE 1588:2008 standard

The system consists of three major components: the *frequency reference*, the *PTP master clock* and the *PTP slave clock(s)*. The detailed explanation of *disciplined system* is not in the scope of this document but in the applications section some examples will be briefly introduced.

2.1 Reference time/frequency source

The primary goal of a clock synchronization system is to transfer the emitted time and frequency of the reference source to the synchronizing nodes. These kinds of devices are not deployable at every site in practice because of their high price (cesium atomic clocks), or their special placement requirements (GPS receivers need free view to the sky). Generally in larger clock distribution networks the direct distribution of the clock signals into every node is impossible, thus a clock hierarchy was made to simplify the needed cabling needs.

The most widespread high–quality reference clock source is the GPS, therefore, I chose it to provide the reference time to the masters of the synchronization system. The interface of the receivers are two–fold, the absolute timing information is available on some sort of serial/parallel signal, mostly transmitted in the form of NMEA sentences and the phase/frequency information is transmitted in a 1PPS signal. The placement of the receiver is crucial; therefore, for indoor applications an extension of these signals should

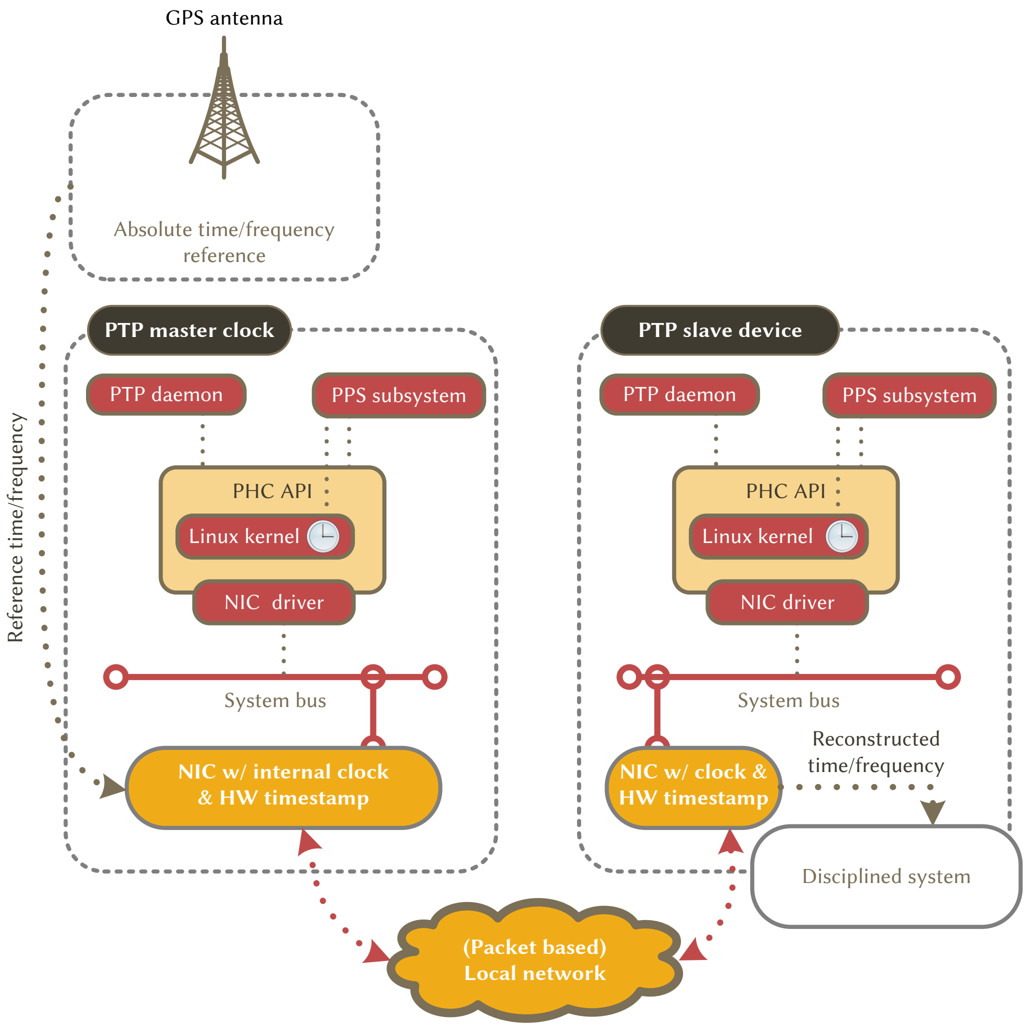


Figure 2.1: The proposed design of the synchronization system

be made in order to connect the PTP master to the remotely placed GPS receiver. To ensure the maximal accuracy, the characteristics of the extension network should be examined too. I chose to feed the PPS signal into the Ethernet adapter to ensure the most accurate reproduction of the incoming clock signal. The clone frequency and the absolute timing informations are combined in a user space utility, which also implements the control loop. The usage of the PHC–API helps to avoid the monolithic design of the control algorithm (hard-wiring it into the driver).

I was given a Garmin 18x–LVC GPS receiver to implement the GPS time/frequency reference services. The receiver outputs the absolute time is on a RS–232 link (encoded into NMEA sentences) and it emits the reference frequency is via 1PPS TTL signal.

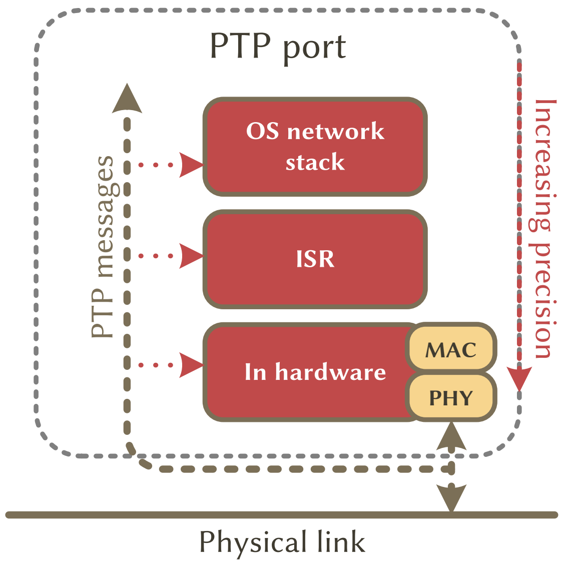


Figure 2.2: Alternatives to acquire message timestamps [15]

2.2 Network interface card

The selection of the NIC decides the attainable accuracy in the synchronization network. There are two major requirements for a sub–µs accuracy: the card should contain a configurable on–board clock, and hardware time stamping capability should be also integrated into it. Of course it should have a working Linux driver, but nowadays it is not a problem for Ethernet adapters. In the proposed design the hardware capabilities are accessed through the kernels `SIOCSHWTSTAMP` ioctl and the *PHC–API* framework.

2.2.1 Time stamping requirements

The available accuracy depends on the short term accuracy of the local oscillators, the implementation of the filtering in the control loop of the software stack, but the most important part is the resolution and jitter of the message timestamps (Fig. 2.2.).

The messages of the PTP protocol can be timestamped in different layers of the operating system. The most widespread solution is to copy the value of kernel's software timer into every message. It is the most obvious, and widely available solution, but the jitter of the timestamps varies between 10-1000 µs, therefore, this method is unacceptable for our needs. The cause of the jitter lies in the representation of the kernel maintained time — it's just an entry in the memory, therefore, it's access is done in non-deterministic time, without upper bound.

Requesting of interrupts at every PTP message has lower jitter than the previously mentioned method, but it performs well only in microcontroller systems where fixed scheduling tables are available, but not in preemptively scheduled environments. The large number of the concurrent tasks & interrupts cause non–deterministic interrupt service periods (tens of microseconds even on a Core 2 Duo configuration).

The autonomous time stamping of packets is the only way to achieve the desired 1 µs accuracy in desktop applications. The independent time stamping cancels out most of the jitters and uncertainties of the measurements. The hardware time stamping can be done in the adapter's MAC and PHY layer, depending on the implementation. The usage of the local onboard clock is common in both methods. The MII, GMII, SGMII, etc. interconnection between them can cause jitter in the time stamps but it's magnitudes smaller than in the previously introduced methods. In my design, the selected Ethernet adapters (Intel i210) has the message filtering and time stamping algorithms implemented in hardware and in its drivers too to decrease the load on the CPU. The hardware timestamps the transmitted messages when the global TX time stamping is enabled, and the packet descriptor has the necessary flags active. The receive logic parses the packets in a configurable way, the hardware supports the IEEE–1588, and the IEEE–1588:2008 standards (the latter is supported over UDP and Ethernet too). The message time stamping point is defined at the last bit of the start–of–frame delimiter of the packet, the local clock is latched at every time stamping point.

2.2.2 Role of the NIC in master clocks

The local clock of network adapter should be disciplined to an external reference source. In my design, the network adapters have the necessary inputs to accurately clone the emitted frequency of the reference, the time information is collected in RS–232 link and also cloned into the on–board clock. To do so the driver and a user–space application should be developed to incorporate the necessary discipline algorithms.

2.2.3 Role of the NIC in slave clocks

In most cases the synchronized local slave clocks should distributed to other devices, therefore, a local clock output should be implemented. With the usage of the network adapter's hardware output capabilities, the frequency information could be accurately reconstructed. The necessary timing information can be supplied in other ways (e.g. RS–232 link).

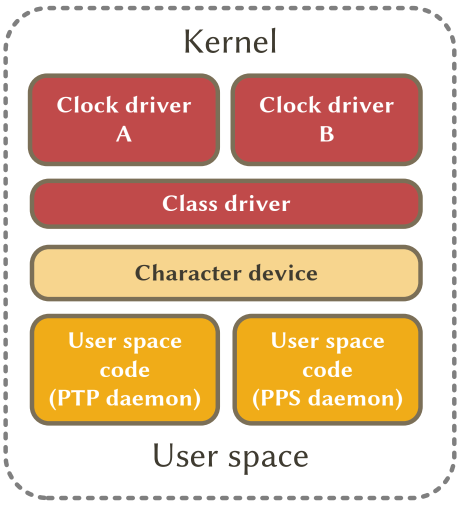


Figure 2.3: The layout of the PHC API [2]

2.3 PHC API

This application programming interface serves as a simplified tool to access and program the on–board clocks on the network adapters (Fig. 2.3.). It was published to the developer community in 2010 and it was merged into the mainline kernel in 2011 (Linux 3.0.0). The existing programs can use the PHC as a time source with small modifications.

The user-space application see the functionality of the clocks via a *character device*. These character devices can be controlled with the previously used POSIX clock handling routines, and the ordinary file manipulation functions. For example, the current time of the clock can be read with the `clock_gettime()` system call, and the `read()`, `open()`, `poll()` and `ioctl()` routines are also usable. The ioctl calls are used for requesting the ancillary clock features which are not covered by the ordinary POSIX clock functions.

The character device is enumerated by the *class driver* which provides interfaces to the kernel driver developers to attach their implementation to the user–space. There are two types of interface functions — the mandatory ones are the:

- query,
- set to arbitrary time,
- adjust with offset,
- and adjust the frequency of the clock.

There's a possibility to implement ancillary features of the clock served through this interface such as:

- program periodic output channels,
- provide a PPS hook for disciplining the local clocks,
- and time stamp external events.

The API supports an interface for implementing programmable periodic timers based on the NIC clock, but currently nobody provided driver for it. With the usage of the ancillary features a high quality master clock can be made with a clean, well–maintained programming interface.

2.4 PTP daemon

The protocol stack implements the specified rules of the synchronization by the IEEE 1588:2008 standard, and implements the necessary unspecified functions too (e.g. disciplines the clocks). In the past I used the *PTPd2* software to provide the needed functionality. In 2011 the *linuxptp* daemon was published, and since the start of the writing of my thesis it evolved into a mature software product.

The PTPd2 daemon was the first open–source implementation of the IEEE 1588 standard, and the base of the first open–source hardware assisted time stamping solutions. The original code supported the software time stamping only, but in 2008 the Intel released a proof of concept hardware time stamping capable fork of the daemon. My previous work was based on the original code base, but some parts included some extra fixes and advancements. Among the main features, the full implementation of ordinary clocks, and the portability & low resource usage are the most important. The lack of boundary clock implementation means that this software can only function in the endpoints of the synchronization network. The low resource usage means fixed point arithmetic in its filters and control loops, thus carefully designed and implemented algorithms are needed to attain the accuracy requirements. The source code was written in C, so it runs on systems where the timestamping in the network stack is implemented. The PTP–API fork of the daemon breaks some of the original features (such as portability), but the clock discipline would work on any adapter with refitted drivers; therefore, it was considered to include in this design. The fork was made by Richard Cochran, the main developer of the linuxptp project; as the linuxptp evolved, the fork rendered obsolete and it's removed from the internet. The usage of an unmaintained software is not preferable in any project, therefore, I chose a different daemon to include against the one stated in the thesis proposal.

The *linuxptp* has a more active development community, and it's goals fit in more in my application; therefore, I chose it as a replacement. It has the following features [3]:

- Supports hardware and software time stamping via the Linux `SO_TIMESTAMPING` socket option.
- Supports the Linux PTP Hardware Clock (PHC) subsystem by using the `clock_gettime` family of calls, including the new `clock_adjtimex` system call.
- Transport over UDP/IPv4, UDP/IPv6, and raw Ethernet (Layer 2).
- Modular design allowing painless addition of new transports and clock servos.
- Implements Boundary Clock (BC) and Ordinary Clock (OC).
- Supports IEEE 802.1AS–2011 in the role of end station.

The software's target platform is Linux only, and relies on floating point operations; therefore, its portability is worse than PTPd's. Compared to the vanilla PTPd, having the first four features are mandatory to enable the design of a high quality PTP master clock. The daemon also implements Boundary Clock functions as opposed to PTPd. It's also written in C language and its structure is well–suited for easy integration of alternative transport layers or clock servos. The support of legacy time stamping and other architectures are not the prime development factors.

2.5 PPS subsystem

The two–way communication between the on–board hardware clock and the kernel is maintained by the PPS subsystem. It is used in two configurations, in the master an externally generated event (the PPS signal of the GPS) is sampled and a PPS event is triggered to discipline the PHC and the kernel clocks. With the software clock discipline, the base frequency of the software control loop is also modified to ensure a more precise correction. It's important that the on–board clock of the NIC should have PTP timescale (most likely), but the Unix time (and the kernel time also) based on UTC timescale, this difference should be known and corrected by the PPS daemon.

In slave systems the PPS events are used to correct the kernel time. In the earlier designs, the kernel clock was synchronized, but in case of hardware time stamping, the local time base of network adapters should have transformed into the time base of the kernel, but the tunable local clocks make this element unnecessary. The kernel clock discipline can be done with alternative methods, for example the `phc2sys` utility bundled in the *linuxptp* package offers an viable implementation with the usage of the kernel PPS subsystem. The separate tuning of the PTP device (network adapter) and the kernel clock is called *two layer synchronization.* Generally the separate PTP clocks have better stability, and accuracy than the kernel clock, therefore, their usage is highly recommended.

2.6 Local network

The local network consists of active and passive elements working together with the synchronization network. To attain the desired accuracy, the network should have a proper segmentation and hierarchy to ensure the lowest possible jitter on the non–deterministic Ethernet carrier. In real world configurations, the usage of Transparent and Boundary clocks are inevitable.

3 | Implementation and usage details

In this chapter, the actual implementation and setup details are discussed in depth. The chosen parts will be examined in every reasonable aspect regarding the system's accuracy.

3.1 Cloning the reference time

The chosen Garmin receiver is suitable for testing the concept of the synchronization system, but it should be noted that its datasheet states that only 1 µs accuracy is guaranteed under a valid position fix (see [16] pp. 20). This upper bound of the error is guaranteed in non–stationary receivers too. It means that an arbitrary node in the PTP network may have more than 1 µs error compared to the ideal GPS reference signal, therefore, the 1 µs accuracy can be guaranteed only within the bounds of synchronization system. There are other commercially available GPS receivers which have better specifications under stationary usage, however the measurements described in the Section 4.2.2 shows that at good reception the desired goal is attainable.

3.1.1 Physical connection of the GPS receiver

In section 1.1 the drawbacks of the GPS was summarized briefly: *"its indoor usage is limited"*. During the construction of the system I have encountered with this problem, thus I needed a solution to connect the test computer in the server room with the remotely placed GPS receiver. In order to achieve the required accuracy, the interconnection has to carry the asynchronous serial and PPS signals from the GPS module for 25 meters. In the laboratory the power outlets are switched of for safety reasons every day, therefore, a remote power supply was also implemented.

The design and implementation of the circuit is made with the help of my advisor to successfully overcome the suboptimal orientation of the server room.

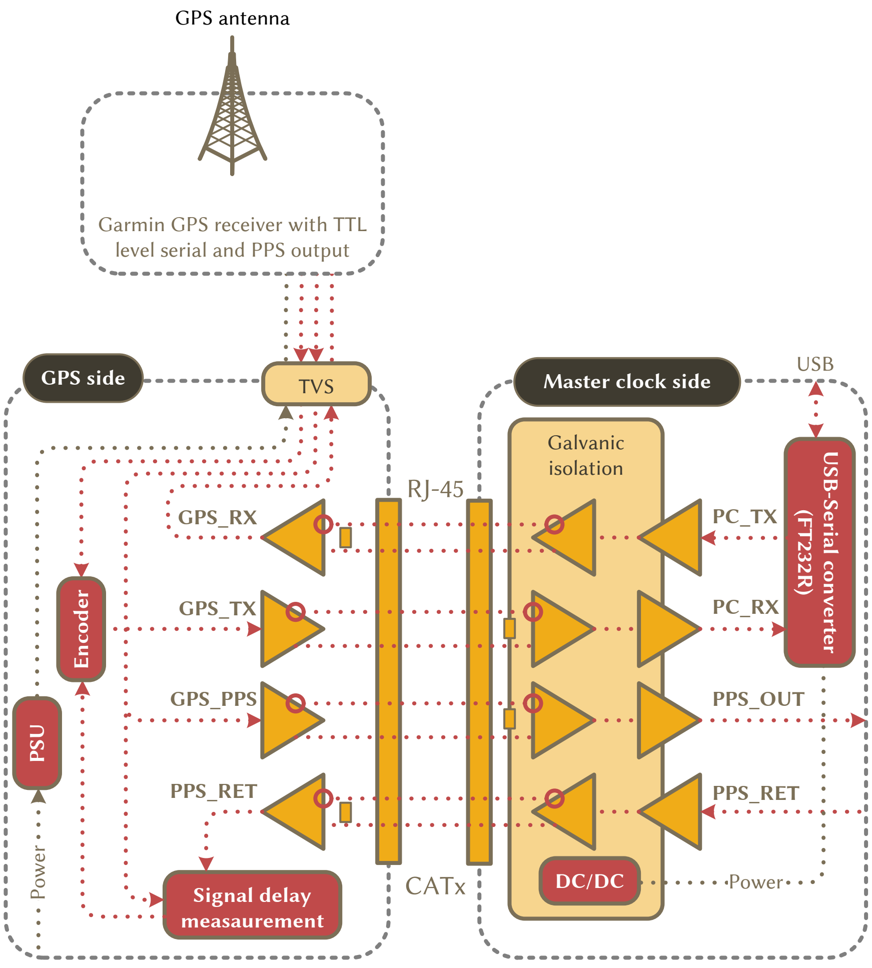


Figure 3.1: Schematic of the GPS extender circuit

The Figure 3.1. depicts the block diagram of the extender circuit. On the GPS side, the receiver is connected to the device through a Transient Voltage Suppressor to ensure the insulation from the lightning hazards. The encoder and the on–line signal delay measurement part is currently unimplemented, their role is to measure the round–trip delay of the PPS signal and transmit it through the RS–422 link with CAT-5e cabling. This measurement cancels out the effect of the cabling delay between the two stations and the delay of the active elements, but it cannot measure the delay between the GPS receiver and the signal converter of that side. The system has active elements in the signal path which degrades the accuracy of the PPS signal and introduces phase and frequency noise. In the final system, a custom NMEA–like sentence will be used to carry the information. To ensure the necessary room to transmit the correction, the speed of the link and the utilization should be properly configured. Right now only offline measurements can be made which are not as accurate as the on–line ones because of the time dependence of temperature and other factors. The equipment is powered by a Power over Ethernet adapter on the GPS side.

On the master clock side, the connection is galvanically isolated from the rest of the system to ensure the lightning protection and electromagnetic compatibility. The PPS signal can be looped back to enable the measurement of the signal delay. The serial link is converted to USB to ensure the easy attachment to modern computers. The equipment also gains its power supply from the USB port. In the operating system the whole system appears as a virtual serial device, which emits the NMEA sentences periodically.

3.1.2 Setting up the serial connection & the gpsd daemon

On the connected serial and 1PPS lines all of the necessary timing information is available to accurately discipline the clock of the PTP master node. The serial line is connected to the serial–USB converter; therefore, the Garmin receiver shows up as a generic USB teletype device (`/dev/ttyUSBx`). Initially its serial port emits the messages with 4800 baud, so the receiving port should be configured to the same speed, otherwise only garbage will appear on the screen! The properties of the receiver (`$PGRMC`) can be acquired by issuing the NMEA commands seen on the listing 3.1).

Listing 3.1: Querying the properties of the GPS module in console

```
onlab@mitpc37:~$ echo -en '$PRGMCE\r\n' > /dev/ttyUSB0
onlab@mitpc37:~$ cat /dev/ttyUSB0
$GPRMC,220553,A,4728.3435,N,01903.6016,E,000.0,000.0,040513,004.0,E*70

$GPGGA,220553,4728.3435,N,01903.6016,E,2,08,1.2,117.0,M,42.0,M,,*4A

$PGRMC,A,00117.0,100,0000000.000,000.000000000,0000,0000,0000,A,5,1,2,24,30.0*7E

^C
```

The section 4.1.3 in the Garmin manual [16] describes the available PPS/serial output settings. Sometimes the receiver seems to be unresponsible to the commands issued from Linux. The only circumvention of this problem was to attach it to a Windows equipped PC and run the Garmin setup program on it (see Appendix A.1). Afterwards the GPS module seems to be working as advertised. I have configured the receiver to emit 50% duty cycle 1PPS and 19200 bps serial output. To finish the setup, one should enable only the most important sentences on the receiver. This task can be done by issuing the `$PGRMO` command to the system (listing 3.2). By default five sentences are enabled on the GPS 18x models, but the application needs only two, the `$GPRMC` and the `$GPGGA` sentences.

Listing 3.2: Enable the necessary NMEA sentences only

```
#!/bin/sh

# Disable all sentences
echo -en '$PGRMO,,2\r\n' > /dev/ttyUSB0
sleep 1

# Enable GPRMC and GPGGA
echo -en '$PGRMO,GPRMC,1\r\n' > /dev/ttyUSB0
sleep 1
echo -en '$PGRMO,GPGGA,1\r\n' > /dev/ttyUSB0
```

If the current settings were read out again from the device successfully, the initial settings are done. To extract the timing messages in a more friendly form, the *gpsd* was used in this setup. The main advantage of this method is that the GPS device can be shared between different applications, and the clock discipline code is free from NMEA message parsing code, because the processing occurs in the gpsd. The client application can connect to the daemon via UNIX sockets and through a shared memory driver too. It should be noted that the SHM driver permits read only access only to the extracted data. It is assumed that the daemon itself is installed somehow in the system — built from the source–code or downloaded with the package manager. If the software is already installed as a system daemon, refer to the documentation about its temporary shutdown.

For the first time it's practical to invoke the daemon with the `-N` switch to hold it in the foreground for debugging purposes, see listing 3.3.

Listing 3.3: A sample run of the gpsd in debug mode

```
onlab@mitpc37:~$ sudo gpsd -b -n -N -D 3 /dev/ttyUSB0
gpsd:INFO: launching (Version 3.4)
gpsd:INFO: listening on port gpsd
gpsd:INFO: NTPD ntpd_link_activate: 1
gpsd:INFO: stashing device /dev/ttyUSB0 at slot 0
gpsd:INFO: opening read-only GPS data source type 3 and at '/dev/ttyUSB0'
gpsd:INFO: speed 19200, 8N1
gpsd:INFO: attempting USB device enumeration.
gpsd:INFO: 0403:6001 (bus 3, device 2)
gpsd:INFO: 1d6b:0002 (bus 1, device 1)
gpsd:INFO: 1d6b:0002 (bus 2, device 1)
gpsd:INFO: 1d6b:0001 (bus 3, device 1)
gpsd:INFO: 1d6b:0001 (bus 4, device 1)
gpsd:INFO: 1d6b:0001 (bus 5, device 1)
gpsd:INFO: 1d6b:0001 (bus 6, device 1)
gpsd:INFO: 1d6b:0001 (bus 7, device 1)
gpsd:INFO: 1d6b:0001 (bus 8, device 1)
gpsd:SHOUT: vendor/product match with 091e:0003 not found
gpsd:INFO: speed 9600, 8O1
gpsd:INFO: speed 19200, 8N1
gpsd:INFO: gpsd_activate(): activated GPS (fd 6)
gpsd:INFO: device /dev/ttyUSB0 activated
gpsd:INFO: running with effective group ID 20
```

```
gpsd:INFO: running with effective user ID 65534
gpsd:INFO: startup at 2013-05-04T23:40:17.000Z (1367710817)
gpsd:INFO: /dev/ttyUSB0 identified as type Generic NMEA (0.543912 sec @ 19200bps)
gpsd:DATA: merge_ddmmyy(040513) sets year 2013
gpsd:DATA: GPRMC: registers fractional time 234018.00
gpsd:DATA: RMC: ddmmyy=040513 hhmmss=234018 lat=47.47 lon=19.06 speed=0.00 track=0.00 mode=2 status=1
gpsd:DATA: GPRMC time is 1367710818.000000 = 2013-05-04T23:40:18.00Z
gpsd:DATA: packet from /dev/ttyUSB0 with {ONLINE|TIME|LATLON|SPEED|TRACK|STATUS|MODE|PACKET|DRIVER|CLEAR}
gpsd:DATA: GPGGA: registers fractional time 234018.00
gpsd:DATA: GGA: hhmmss=234018 lat=47.47 lon=19.06 alt=119.40 mode=3 status=2
gpsd:DATA: GPGGA time is 1367710818.000000 = 2013-05-04T23:40:18.00Z
gpsd:DATA: packet from /dev/ttyUSB0 with {ONLINE|TIME|LATLON|ALTITUDE|STATUS|MODE|PACKET}
gpsd:DATA: merge_ddmmyy(040513) sets year 2013
^C
```

After the successful enumeration of the device (matching the type and speed of the device), the daemon starts to collect the information from the receiver. The CC of the GPS is steered to the UTC, so the transmitted timestrings are in UTC scale, thus the time difference from the PTP timescale should be corrected.

If the proper configuration is found, it's safe to start the gpsd in the background as a daemon, simply omit the `-D` and `-N` switches. To configure the daemon as a system service please refer to the documentation of your distribution. After the procedure, the gpsd is ready to transmit the collected information to the time sync utilities to discipline the clock of the network adapter.

3.1.3 Disciplining the time and frequency on network adapters

I chose to implement the final system with an Intel i210 Ethernet Server Adapter in the role of the network adapter [17] (Fig. 3.2.). The i210 brought innovations into its clock subsystem, the most important was the widened length clock registers (compared to the i350 series of adapters).

First of all the comparison of the underlying clock architecture will be discussed (see Fig. 3.5.). The cards feature a controllable local clock, stored in the `SYSTIM` register group. The effective length of the clock register set is 64 + 32 bits which means that the i210's clock wraps around every 130 years or in the 92 years if we store the Unix timestamp in it. I chose to store the modified Unix time (with the correction of the UTC–TAI) in the registers directly, because it simplifies the access and manipulation of the onboard time both in speed, efficiency and maintainability. The adapter supports Gigabit Ethernet over copper media standard (1000BASE–T) thus the fundamental cycle time is 8 ns on the adapter derived from its 125 MHz clock. At every base clock tick, the clock registers are refreshed, with the addition of the period time, and the value of the `TIMINCA` register (see Eq. 3.1). The `TIMINCA` register is used for frequency adjustment purposes, the controller

Figure 3.2: The Intel i210-T1 Ethernet Adapter

can slew the clock with the stored fraction of time (2^{-32} ns) at every period.

$$SYSTIM[k+1] = SYSTIM[k] + 8\ ns \pm TIMINCA \tag{3.1}$$

The card has 4 *Software Definable Pins* (SDP), which may be mapped to the clock functions. To discipline the on–board clock, it's possible to map one of them to the Auxiliary Time Stamping registers (`AUXSTMP0` and `AUXSTMP1`) and sample an external signal (1PPS in my case). The card is programmed and controlled by the *igb* driver; therefore, I have extended the functionality of it. The initialization of the sampling is on Figure 3.3..

The timestamping of an external 1PPS signal directly gives the deviation from the start of the reference second. This error signal is used to feed a PI servo to control the frequency of the clock. The PTP timescale is monotonically increasing without any inserted leap seconds, thus we need to initialize the clock on every startup, and after that the frequency control is enough to keep the accuracy within specifications.

The enabled time syncing functions in the recent drivers need enabled PHC–API in the Linux kernel. In most distributions it's not included by default, and custom compiled kernel needed to work (for Ubuntu see [18]).

I have successfully extended the igb driver to sample the external events. The time stamping mechanism is triggered by rising and falling edges of the incoming signal. The only way to distinguish between them is to read back the value of the sampled pin in the ISR. To lower the interrupt servicing delay, the ISR itself schedules another task to do the work. The delay between the edges is in the magnitude of ten to hundred milliseconds,

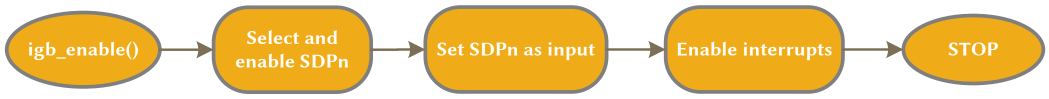


Figure 3.3: Initialization of pin sampling

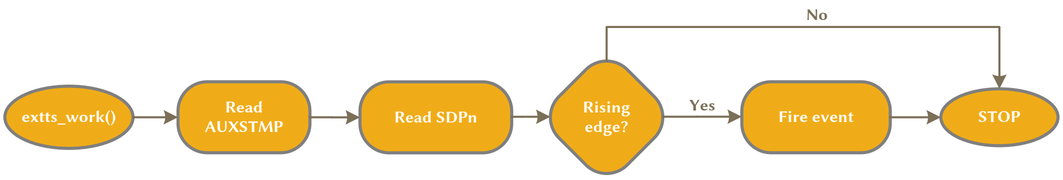


Figure 3.4: Processing of the edges

and the whole process should not take more than few milliseconds. Both hardware can hold only one time stamp, but in most cases, the samples are processed fast enough that it won't be a problem. The processing of the time stamps is depicted byFigure 3.4. The readout of the timestaps is done through the raw read of the `AUXSTMP` registers. The sample of the SDP pin is necessary to decide the direction of the sampled edge (rising/falling). The readout of the time stamps are necessary in every cases, because the AUXSTMP registers are latched and won't receive any new event stamp until they are read out. At the end the routine fires an event which can be polled by the user–space programs through the PHC–API.

The driver modifications are also applicable to the multi–port i350 family of network adapters [19]. There are few minor differences between the clock architecture of the adapters, such as the onboard clock of the i350 is just 40 bits long. This limitation is circumvented with the usage of the clock infrastructure in the kernel which supports arbitrary long time handling with finite width counters. The time handling in the driver routines in this case always includes a time conversion call may lower the accuracy by negligible amount. The multi–port i350 silicon contains 2 or 4 *independent* onboard clocks, which should be synchronized by the user in some applications (TC, BC).

I have developed an user–space utility called `ts2phc`, which is a modified version of the `phc2sys` utility included with the linuxptp. The software feeds the absolute time into the PHC's clock at startup and the included servo controls the frequency of the clock. The absolute time is queried through the gpsd's SHM driver. The driver of the adapter doesn't arm the interrupts of the card unless it has an active IP address; therefore, the utility needs an active, configured adapter, otherwise the software does not function. The console output of the software is shown in listing 3.4.

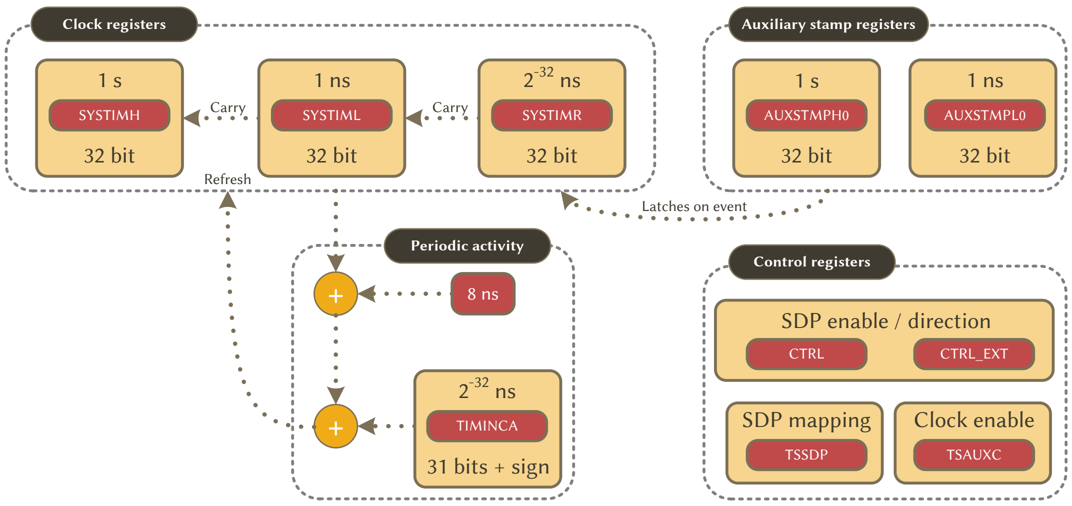


Figure 3.5: Clock internals of the i210 adapter

Listing 3.4: A sample run of the ts2phc

```
onlab@mitpc37:~$ sudo ./ts2phc -d /dev/ptp0 -g -t 35
fix.time: 1367763813    extts 199993376 s1 1367763794.199993376 drift 0.00
fix.time: 1367763813    extts 238925608 s2 1367763849.238925608 drift 0.00
fix.time: 1367763814    extts 238902968 s3 1367763850.238902968 drift 0.00
fix.time: 1367763815    extts 238880320 s4 1367763851.238880320 drift 19443472.00
fix.time: 1367763816    extts -19467954 s4 1367763851.980532046 drift 13603085.80
fix.time: 1367763817    extts -19467308 s4 1367763852.980532692 drift 13603085.80
fix.time: 1367763818    extts -18977997 s4 1367763853.981022003 drift 13603085.80
fix.time: 1367763819    extts -18488649 s4 1367763854.981511351 drift 13603085.80
fix.time: 1367763820    extts -17999308 s4 1367763855.982000692 drift 13603085.80
fix.time: 1367763821    extts -17509960 s4 1367763856.982490040 drift 13603085.80
fix.time: 1367763822    extts -17020620 s4 1367763857.982979380 drift 13603085.80
^C
onlab@mitpc37:~$
```

The output shows the time queried from the GPS (column fix.time), the error from the start of the second (column extts), the state of the servo (column s[N]), the timestamp read out from the network adapter, and finally the drift. The program can work without the gpsd (omit `-g`) but in that case the user should supply the clock's starting value in some other way. The `-t <number>` switch supplies the offset of UTC–TAI timescales at the startup. The program supports the delay compensation of the sampled signal, supports multiple event sources (on the i210 I implemented both channels). After the first batch of initialization it starts the servo loop. In the loop the external event source is opened, then it is read out peridically, the clock is disciplined in an infinite loop. The software supports accurate synchronization to a stable external reference, see Figure 4.5..

3.2 Distribution of time

In the previous section we discussed the implementation details of the PTP master clock as a slave to the reference source. In this section I will cover the functionality needed to provide the reference time to other nodes.

3.2.1 PTP services using linuxptp

The linuxptp daemon is used to realize the PTP services in the master clock. In the measurement network only it's Ordinary Clock functions were used, the message interchange was done over IPv4/UDP. This decision was made because the interconnecting equipment only supports Transparent Clock services over UDP, so it was the only viable option for the direct comparison for the measurement of complex topologies. The linuxptp software is mostly distributed in source code form (see Appendix A.1), but in Fedora distributions precompiled packages are also available. If the software compiled successfully it is ready to use.

Four executables are supplied with the package: the `hwstamp_ctl` is a utility to change the applied message time stamp filter on the network adapter. It's useful only for debugging purposes as the daemon sets the necessary adapter capabilities independently.
The `phc2sys` is a tool to discipline an arbitrary POSIX compatible clock (such as the local kernel clock) with respect to the PHC's own clock or to any PPS source. It is possible to choose between the event source, one can directly read out the PHC, or rely on the PPS signals. The direct readout of the onboard clock may cause undesired race conditions, which aren't handled in the API or the igb driver as yet, therefore, it's recommended to implement and use the PPS functionality. The PI constants of the included controller can also be changed in the command–line.
The `ptp4l` implements the PTP daemon itself, it supports the ordinary and boundary clock roles. It needs the used interface as an input argument to start successfully, but the majority of the configuration options are read from a text file. If it is started with the `-m` switch (see listing 3.5., it logs to the `stdout`, otherwise it writes its debugging messages only to the system log. The verbosity of the messages are configurable, in a production setup only the most important events/errors are logged, but one can write out the content of every message.

Listing 3.5: A sample run of the ptp4l (slave state)

```
onlab@mitpc37:~$ sudo ./ptp4l -f default.cfg -i eth7 -p /dev/ptp0 -m
ptp4l[162683.928]: selected /dev/ptp0 as PTP clock
ptp4l[162683.929]: failed to read out the clock frequency adjustment: Operation
not supported
ptp4l[162683.929]: port 1: get_ts_info not supported
ptp4l[162683.930]: driver changed our HWTSTAMP options
ptp4l[162683.931]: tx_type   1 not 1
ptp4l[162683.931]: rx_filter 1 not 12
ptp4l[162683.931]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[162683.931]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[162684.748]: port 1: new foreign master 0050c2.fffe.d28dfc-1
ptp4l[162688.898]: selected best master clock 0050c2.fffe.d28dfc
ptp4l[162688.898]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[162689.988]: port 1: minimum delay request interval 2^3
ptp4l[162691.037]: master offset       -636 s0 adj      +0 path delay       1686
ptp4l[162692.038]: master offset       -679 s1 adj     -43 path delay       1686
ptp4l[162693.188]: master offset     -30508 s2 adj  -30551 path delay       1686
ptp4l[162693.188]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[162694.198]: master offset     -22593 s2 adj  -31788 path delay       1686
ptp4l[162695.198]: master offset     -13494 s2 adj  -29467 path delay       1686
ptp4l[162696.339]: master offset      -5718 s2 adj  -25739 path delay       1686
ptp4l[162697.348]: master offset      -2611 s2 adj  -24348 path delay       1686
ptp4l[162698.348]: master offset       -990 s2 adj  -23510 path delay       1686
ptp4l[162699.439]: master offset        -62 s2 adj  -22879 path delay       1686
ptp4l[162700.458]: master offset         95 s2 adj  -22741 path delay       1686
ptp4l[162701.458]: master offset        164 s2 adj  -22643 path delay       1686
ptp4l[162702.548]: master offset        163 s2 adj  -22595 path delay       1686
ptp4l[162703.569]: master offset         93 s2 adj  -22616 path delay       1686
ptp4l[162704.568]: master offset        -18 s2 adj  -22699 path delay       1686
ptp4l[162705.719]: master offset         20 s2 adj  -22667 path delay       1686
ptp4l[162706.738]: master offset         47 s2 adj  -22634 path delay       1686
ptp4l[162707.739]: master offset         10 s2 adj  -22657 path delay       1686
ptp4l[162708.889]: master offset        -46 s2 adj  -22710 path delay       1686
ptp4l[162709.899]: master offset        -47 s2 adj  -22724 path delay       1686
ptp4l[162710.899]: master offset         45 s2 adj  -22646 path delay       1686
ptp4l[162712.049]: master offset          7 s2 adj  -22671 path delay       1686
ptp4l[162713.069]: master offset        -27 s2 adj  -22703 path delay       1686
ptp4l[162714.069]: master offset        -12 s2 adj  -22696 path delay       1686
ptp4l[162715.219]: master offset         22 s2 adj  -22666 path delay       1686
ptp4l[162716.229]: master offset         60 s2 adj  -22621 path delay       1686
ptp4l[162717.229]: master offset        -32 s2 adj  -22695 path delay       1686
^Cptp4l[162717.953]: caught signal 2
ptp4l[162717.953]: caught signal 2
onlab@mitpc37:~$
```

The user can configure the BMC fields in the configuration file has the most important fields, such as `clockClass`, `clockAccuracy`. When the internal clock is adjusted to the external GPS, the clock class is set to 6_d which denotes frequency traceable time source. If the GPS reception is unavailable it should be set to 7_d but the automatic setting is currently unimplemented. The clock's accuracy is set the 26_h, which means that the clock provides the time with 1 µs accuracy (worst case estimate). I haven't measured the Allan deviation of the clock so I left the default `offsetScaledLogVariance` value in the

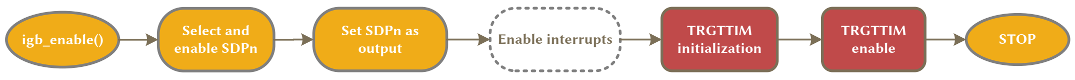


Figure 3.6: The initialization of the 1PPS output

settings. The field `free_running` in a configuration file is used in certain measurements for example to measue the asymmetries in the network. The measured asymmetry afterwards can be mitigated in the servo with the usage of the `delayAsymmetry` field.

The linuxptp daemon is configurable run–time to some extent, and we can control the daemon's operation and it's datasets with the provided `pmc` tool. I have not looked into the depths of the tool, but it's usage is inevitable to make the failover (e.g. end of GPS reception) automatic.

3.2.2 Supporting legacy equipment

The support of non–PTP capable equipment is inevitable to make the system usable in practice. The clock information is should be distributed in at least two different ways:

1. The absolute clock information over a parallel/serial protocol,
2. the phase and frequency information via one–wire connection.

The absolute timing may implemented in an absolutely software defined solution, for example a program can read the PHC repeatedly and copy the value of it to the serial port. The phase information shall be generated by the network adapter to ensure maximal accuracy of the system. The i210 adapter supports the generation of 1PPS signals in a completely autonomous way (only for 50% duty cycle).

I have developed the necessary routines to enable the 1PPS and high frequency output of the network adapter in the igb driver. On the i210 both functionality is provided by the `FREQOUT` registers (on two independent channels) [17]. The i210 boards support the range 16 ns – 70 ms and 250, 500 and 1000 ms period length on the clock output. I have also made a small utility to program the outputs of the card instead of the already provided sysfs file hook, it's called `perpps`. The initialization the output in the user space is requested via the `ioctl` call to the clock. In the driver the `igb_enable()` captures the request and does the work depicted on Figure 3.6. The selection and setting the output pins is the similar to as described in section 3.1.3.

The i210 hardware supports a periodic interrupt with the `SYSWARP` interrupt source, it always activated when the `SYSTIML` register overflows. The `SYSTIML` overflow period

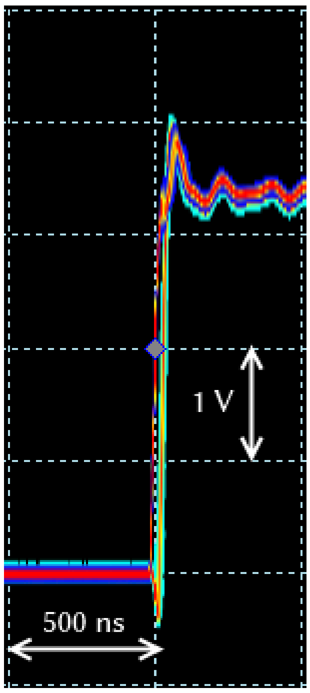


Figure 3.7: The accuracy of the output wrt. external signal

time corresponds to one second measured on the local clock. I've bound the software PPS event generation to this interrupt. The first edge of the output is adjusted to the start of the next full second of the onboard clock with the usage of the `TRGTTIM` registers. In my implementation it is done by setting the timer to half second earlier (than the next full second), because the hardware always initializes the output to zero when the starting condition reached.

The signal generation also works on the i350 adapters; however, the algorithm is a different [19]. The 1PPS output is generated with the periodic reprogramming of the `TRGTTIM` registers, which are programmed to invert the output when the onboard clock reaches the register value. It is the way to generate 1PPS signal also on the i210's, if the default 50% duty cycle is not appropriate for the user's needs. It's advised to use the `TRGTTIM` registers to generate at most 50 Hz signals, because the periodic ISR procedure's execution time has no known upper bound in non–realtime environment. To generate high frequency outputs on the i350 the I have developed the programming interface of the `FREQOUT` registers to generate two independent phase–right output signals. The i350 board supports period times between 16 ns – 4080 ns.

4 | Measurements and validation

The accuracy of the synchronization system can only specified after series of measurements. After the introduction of the test equipment and the initial calibration steps, the test methods will be introduced. This chapter concludes in a detailed explanation of the measurement records.

4.1 Testing equipment & configuration

4.1.1 Reference clock for calibration & validation

In my test setup a Meinberg LANTIME M600 GPS PTPv2 reference clock (Fig. 4.1.) were used to validate the master clock's accuracy. It can synchronize itself to GPS, PTP or NTP time sources and it can propagate the timing informations over NTP and PTP protocols (see [20]) or via legacy outputs. The equipment has serial, PPS, Pulse Per Minute, 10 MHz, Time Code and an arbitrary frequency synthesizer output which covers almost every functionality needed by legacy equipments. In the following measurements the integrated OCXO based clock was disciplined by the GPS receiver. The crystal's insulation from the ambient temperature (hence the name *oven controlled*) improves the internal clock's holdover capabilities, when the GPS reception is unavailable. The GPS receiver of the equipment uses downmixing in its antenna; therefore, the downmixed GPS signal can be transmitted over cheap (RG–58) coaxial cabling. The reference clock can be managed through a dedicated network interface and its PTP functions are served through a separate Ethernet port and has a user–friendly web interface for configuration.

Figure 4.1: Meinberg M600 GPS/PTP reference clock

4.1.2 Nodes for testing and remote development

The majority of the development and measurements were done through remote connection because the server room — where the synchronization test setup is located — is an inadequate place to work for longer time. The development of the system involved changes in the `igb` driver so the risk of an unintentional kernel panic was high. To circumvent these obstacles the development system ran on top of a VMware ESXi hypervisor which was protected from the hangups of the running virtual machine used for testing. The tested network adapter was assigned to the VM with I/O virtualization, therefore, the programming of the device was completely equivalent to a non–virtualized one. The development network can be seen on Figure 4.2..

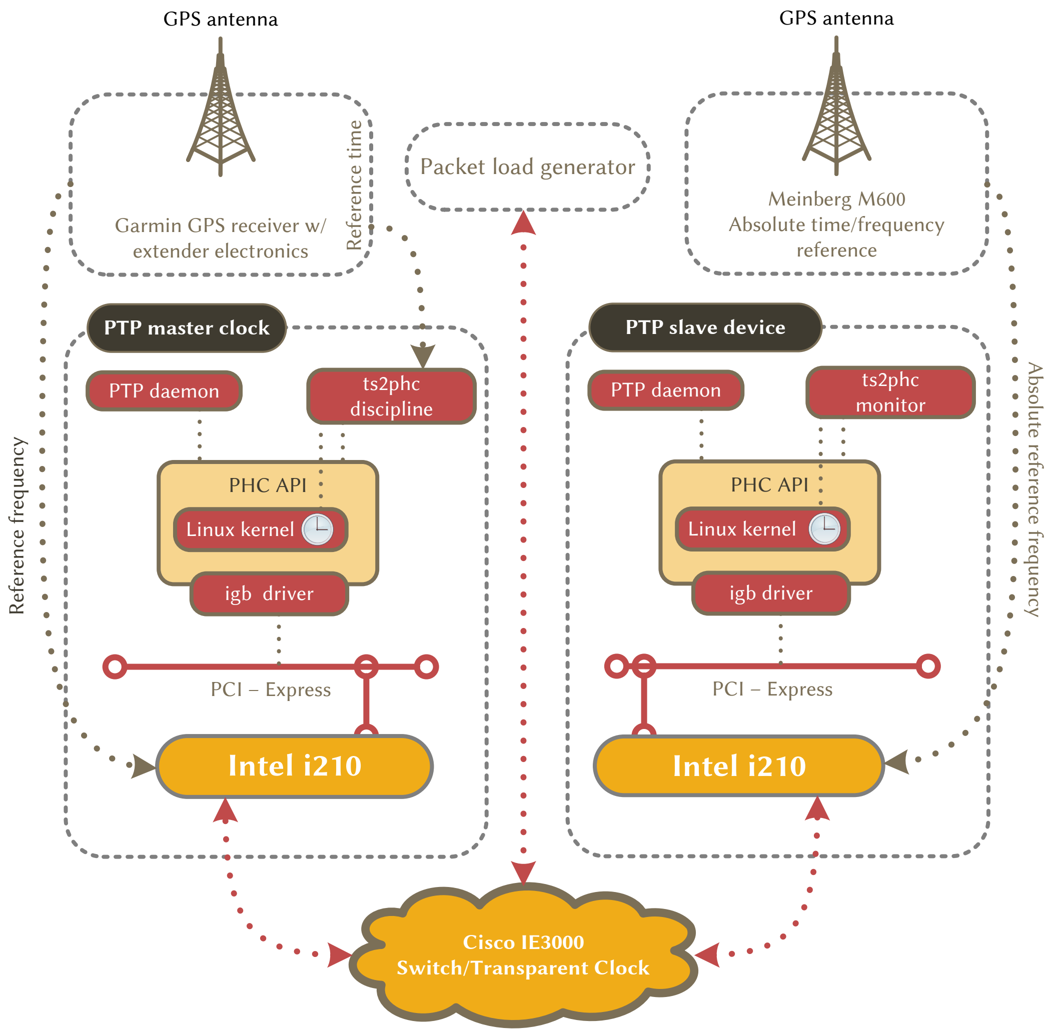


Figure 4.2: Development/measurement network

The Figure consists of the concrete elements in the previous chapters (see Figure 2.1. for comparison). The *packet load generator* was a VM on the development system, the packet generator was the `iperf` tool. It's a client/server software which generates dummy packets on the client and discards them on the server. I used the artifically generated network traffic to demonstrate of the effects of the high network load on the system's accuracy. The tests were conducted on Ubuntu Linux 12.04 with kernel version 3.5.0-ptp. This kernel is not part of the default distribution nor the common package repositories, therefore, a small walkthrough is included in Appendix **??** to help the reader to reproduce the same measurement system.

4.1.3 Other active elements

For the multi–slave measurements the nodes were connected via a Cisco IE3000 Industrial Ethernet switch. The switch supports the 100BASE–TX standard communication on 8 ports and 1000BASE–T on 2 ports and it can act as a PTP Transparent Clock or a PTP Boundary Clock over UDP. I did measurements with enabled and disabled transparent clock modes to show the change of accuracy under heavy load. The basics of the transparent clocks are described on page 7.

Figure 4.3: Cisco IE3000 (4 port version)

Initially, the output signals of the synchronization system were measured with a Picoscope. The control and visualization software of the device ran on the `weasel` virtualization host in a separate virtual machine. The scope's digital phosphore mode were used as an independent tool to visualize long–term effects. The instrument used in the measurements has 1 gigasample/s sampling speed on one channel or 500 megasample/s on two, and the bandwidth on of the inputs (250 MHz) is high enough to correctly sample the square wave signals. Therefore, the instrument is suitable for the measurements presented in this paper.

4.2 Measurement methodologies & results

4.2.1 Measurement methods

In the initial measurements I checked the synchronization quality of the signals with an oscilloscope to make sure that the software reported offset values are valid. This method is useful because the visualization of the signals is made by an autonomous equipment. I also wanted to show the histograms of the measurements which is not supported by the oscilloscope, so the absolute reference or the Garmin receiver's signal was fed into the i210 adapter. When the synchronization of the cards were done by the linuxptp daemon, the adapter generated timestamps from the incoming signal with usage of the `ts2phc` utility. In this case the timestamps show the error with respect to the start every GPS second and they show the accuracy of the system compared to a reference quality 1PPS signal. When the card was synchronized to the absolute reference 1PPS/serial time information, I was able to use it as a timestamp unit to measure another signal too (presented in section 4.2.2). With this measurement method I converted the device under test into a measurement device.

4.2.2 Calibration process

Calibration for the Tx/Rx asymmetry

The PTP algorithm assumes that the link delay is symmetrical in the physical network. Unfortunately, in real–world scenarios the majority of the network adapters have different signal propagation times on their transmit and receive paths. These kinds of asymmetries are near to the magnitude of our desired accuracy goal; therefore, the measurement and compensation of them are inevitable.

The Tx/Rx asymmetry measurement test setup is pictured on Figure 4.4., the devices were connected directly. The combination of the serial and 1PPS timing information was fed into the `ts2phc` tool which disciplined the clock of the PTP master node and the linuxptp daemon ran in simulation mode (without any clock correction applied). The output of the linuxptp was processed and presented on Figure 4.5. For the measurements the Tx

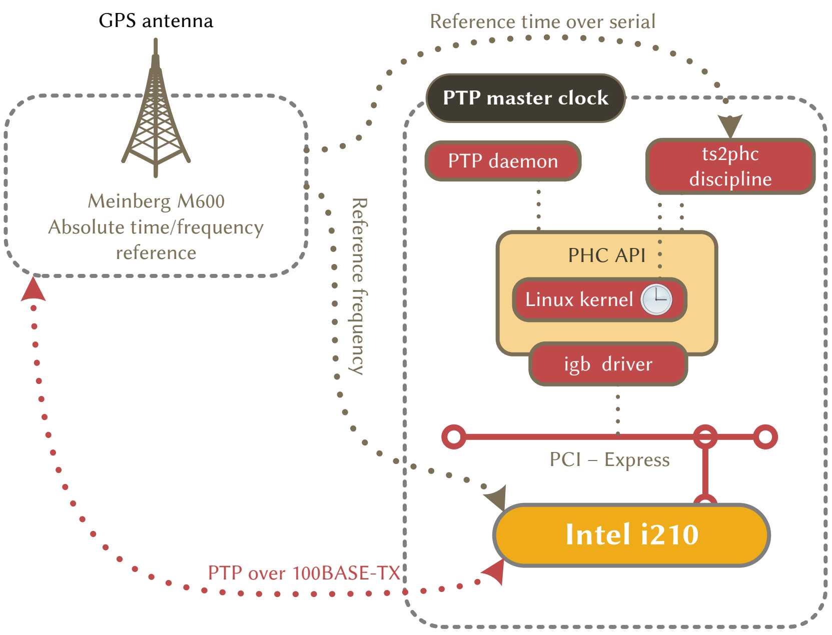


Figure 4.4: The TX/RX asymmetry measurement setup

and Rx signal path of the Meinberg reference clock is assumed to be balanced within ± 20 ns. The datasheet of the i210 adapter states that the difference between the two signal routes is in the magnitude of microseconds, therefore, one can safely abandon the reference clock's asymmetry in the calculations.

With this information the absolute asymmetry of the i210 adapter is measured on 100 MBit/s Ethernet link, but it is only valid on 100BASE–TX Ethernet connections for this adapter! The computed Tx/Rx difference in the next measurements is compensated within the PTP daemon with the mean of the measurements. For more complex topologies every active element of the network should be calibrated to attain the 1 μs accuracy requirement.

After the initial calculations, I tested the calculated asymmetry value of the card with back–to –back measurements. I used a similar setup like on Figure 4.4. but the clock was disciplined over PTP by the Meinberg as master and I used the 1PPS output of the master clock to measure the offset between them.

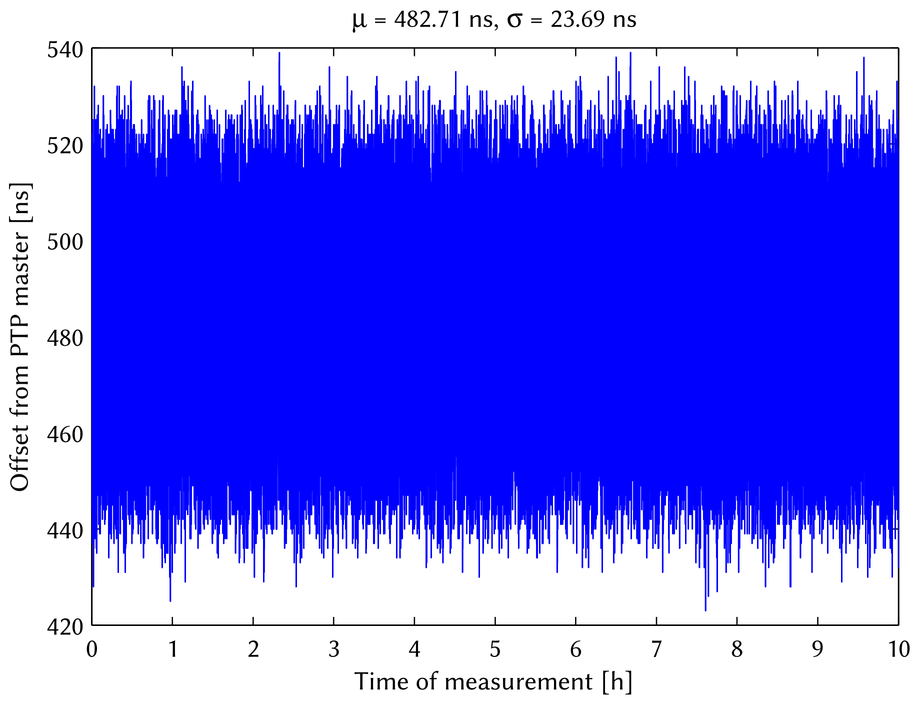


Figure 4.5: The TX/RX asymmetry (extracted from ptp4l output)

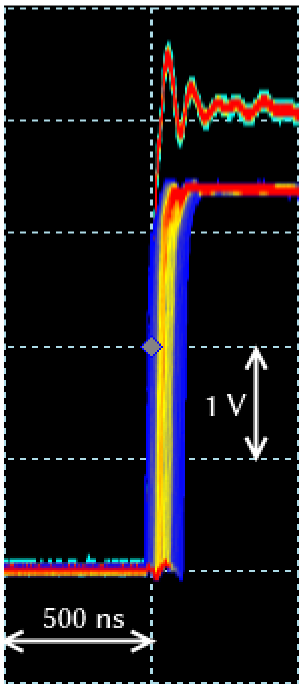
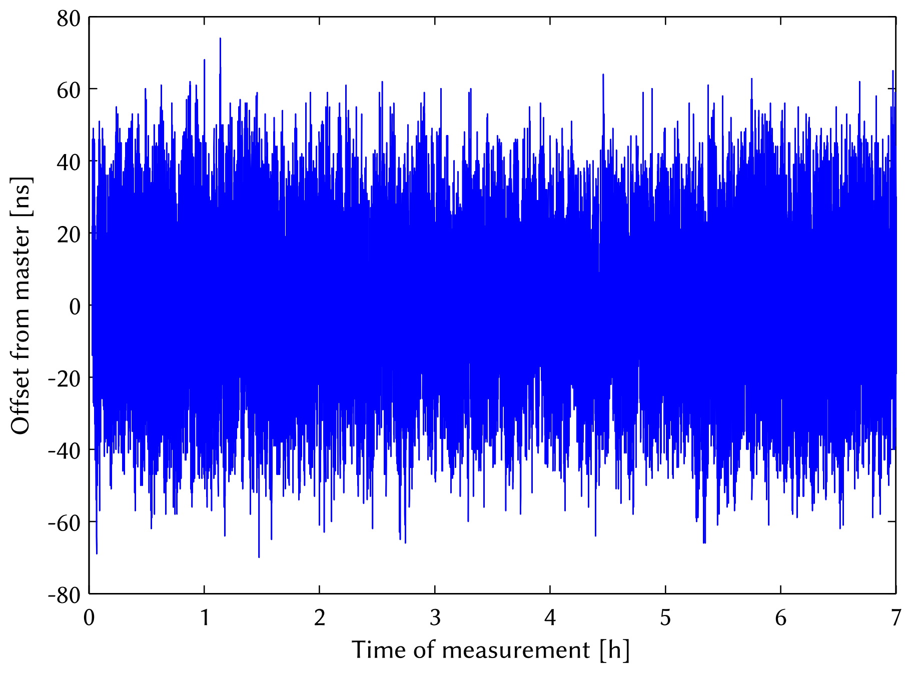


Figure 4.6: The offset from the **absolute reference** with 1PPS discipline on the i210 master measured with Picoscope with applied correction



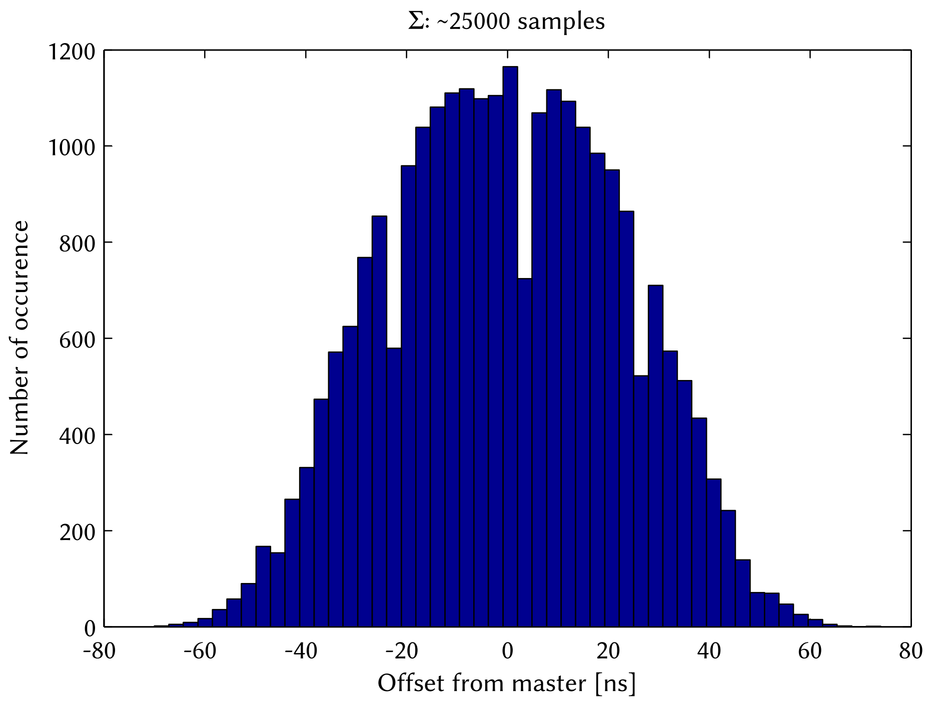


Figure 4.7: The offset of the slave from the master after applying the correction (time series & histogram)

Calibration for the GPS signal delay

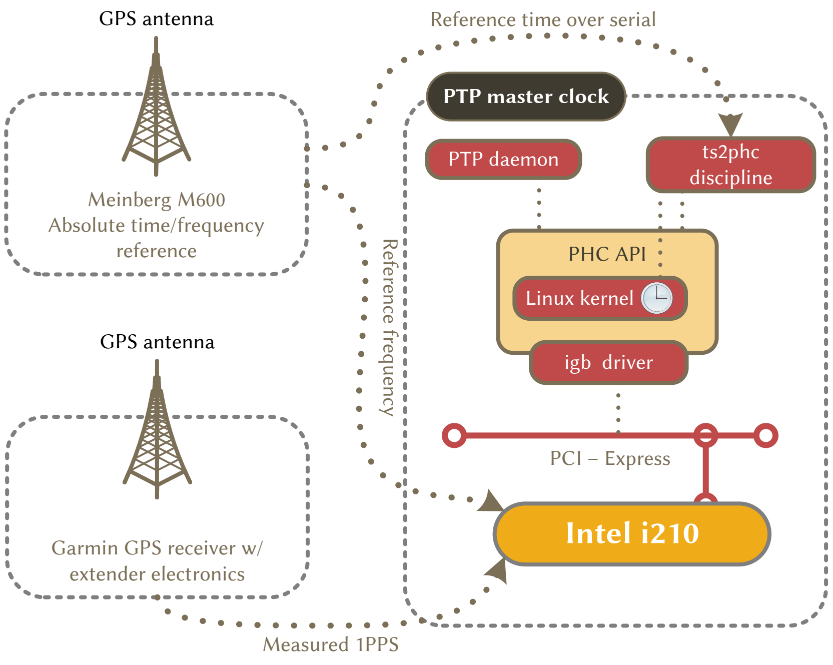


Figure 4.8: The measurement setup of the Garmin receiver's delay

The GPS delay measurement test setup is depicted on Figure 4.8. The Figure 4.9. shows the delay of the PPS signal routed through the GPS extender compared to PPS output signal of the reference clock. This signal delay consists of two components:

1. The uncertanity of the PPS sigal from the GPS receiver,
2. and the delay of the signal converter electronics.

In this measurement setup only the cumulated error of these two components is measureable.

On the time diagram (Figure 4.9.) one can discover that it doesn't show white–noise properties. The Garmin receiver is only guaranteed to be accurate within 1 us and the initial measurements showed that the GPS extender circuit introduces 200–250 ns delay into the system. In more precise application it is mandatory to examine the causes of the wander of the signal, but now I approximate the signal with its mean value. In the following measurements I compensated with the former value, it is adjustable with the `ts2phc` utility.

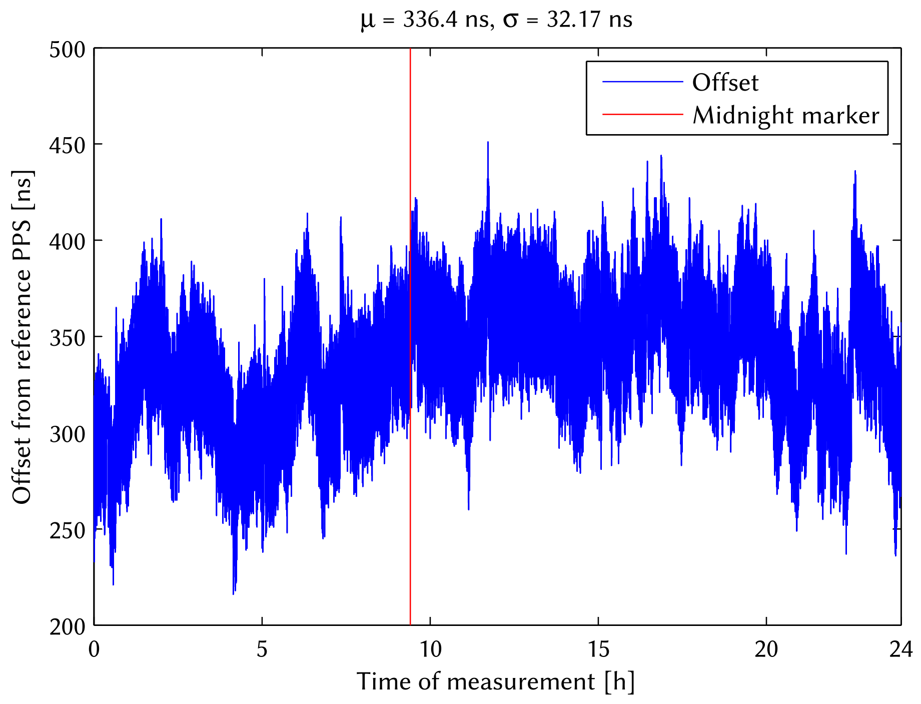


Figure 4.9: The delay of the Garmin receiver and the signal adapter

Results of the calibration & the initialization

First of all I present the initial conditions of the system. After a cold–boot the system with a properly configured network adapter is capable to work as a PTP master or slave. On The Figure 4.10. one can see that the clock of the network adapter runs on the frequency of the onboard oscillator. The Ethernet standard requires quartz oscillator with the accuracy of ± 50 ppm over the entire operating temperature range, we can see that this particular board meets the required specification. I didn't represent it on the figure, but there is a 35 second difference between the first 15 timestamps and the rest, because the `igb` driver initializes the adapter to the UTC timescale, but the `ptp4l` daemon sets the clock to the PTP timescale on the nodes which differed 35 seconds at the time of the measurement.

To test the accuracy of the GPS receiver and asymmetry calculations, the setup of the system was the same as depicted on Figure 4.8., but the timing information was pulled from the Garmin receiver also. The Meinberg reference clock and the DUT was connected back–to-back with each other to eliminate the asymmetry of any kind of active network element. I used the ptp4l daemon as an observer, the `ts2phc` tool disciplined the i210's onboard clock. The measurement records made with Picoscope are depicted on Figure 4.11. and the time stamped values on Figure 4.12., with the corresponding histogram.

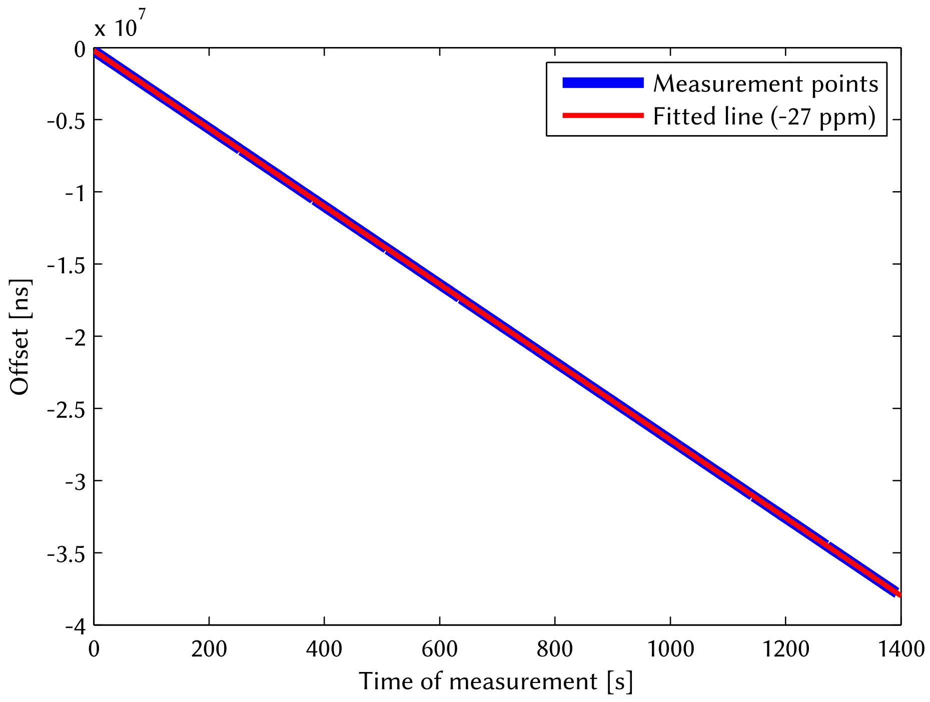


Figure 4.10: The initial drift of the system

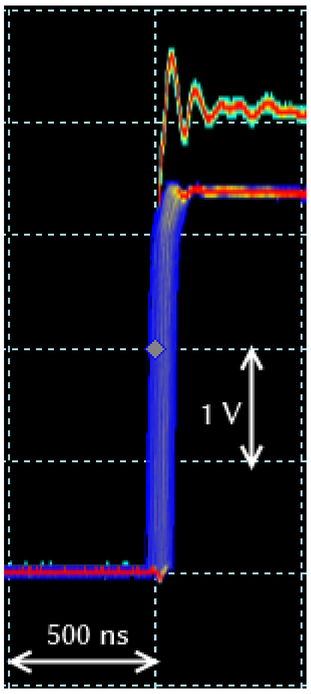
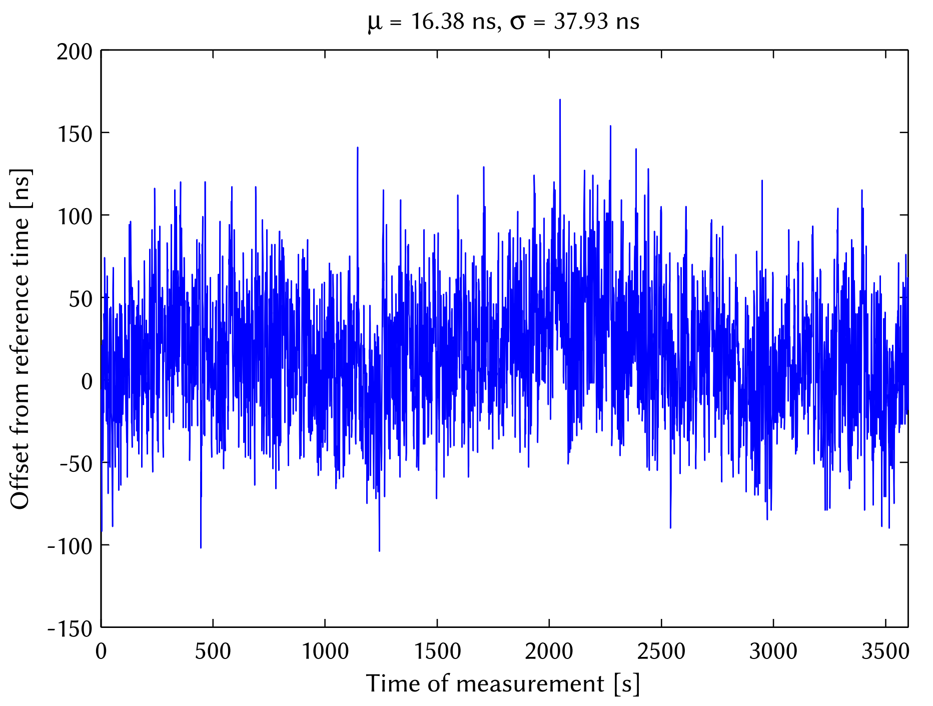


Figure 4.11: The offset from the **absolute reference** with direct connection to the i210 master measured with Picoscope



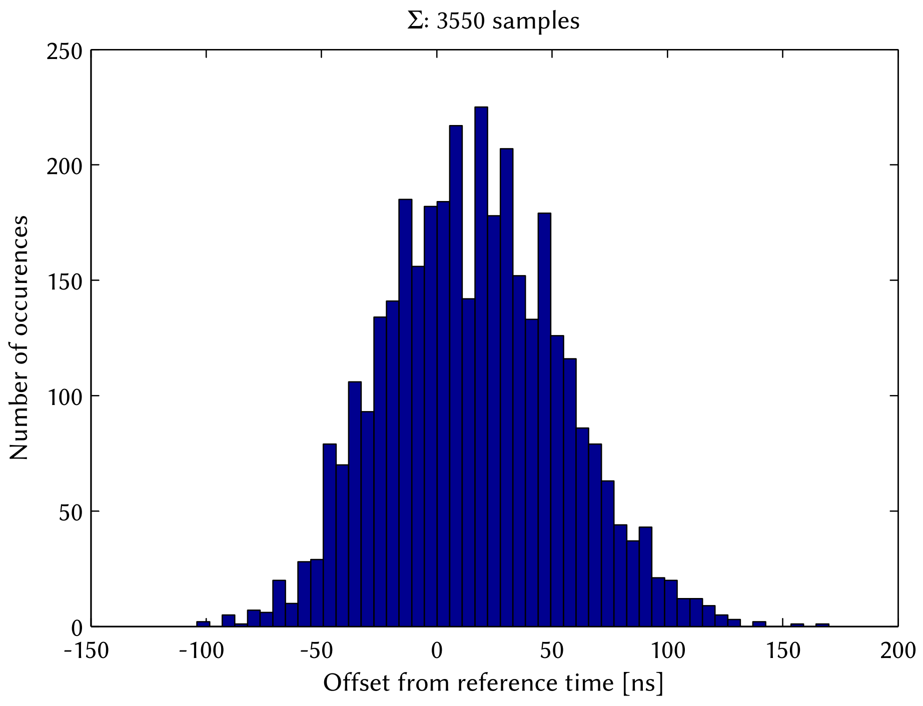


Figure 4.12: The offset from the **absolute reference** with direct connection to the i210 master (time series & histogram)

If the reception of the GPS signal is jammed by some atmospheric disturbance, or any other unforeseeable event, the system may stop the disciplining of the clock and try to work without any external reference. It is a currently unimplemented feature, but it would be a fine error handling scheme. The Figure 4.13. depicts a situation where the last applied clock frequency modifier regulates the flow of the time on the adapter. The initial -27 ppm error is reduced three magnitudes (36 ppb) in this case, but it may have better or worse performance based on the last applied clock modifier. The examination of these kind of nomad synchronization algorithms on Ethernet adapters is outside of the scope of this document.

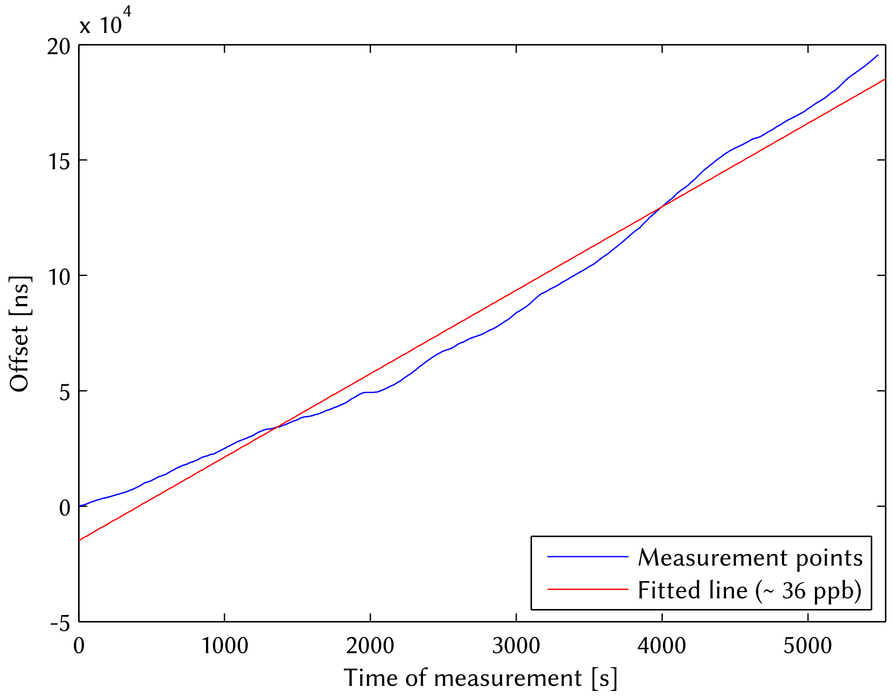


Figure 4.13: An example holdover drift of the system

4.2.3 Measurements with an active network element

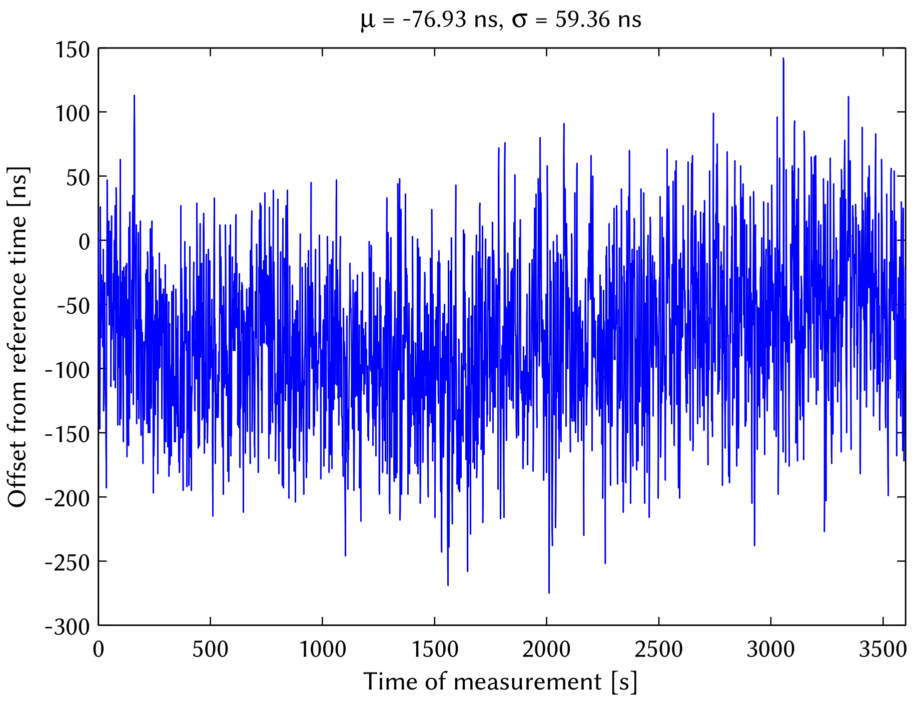
This section presents the measurements from the most common scenarios of PTP systems. The usage of transparent clocks is widespread in industrial applications, especially in freshly deployed systems. However, there are millions of already installed systems where the network infrastructure does not have the transparent clock functions. The following figures demonstrate the quality difference between the various equipment settings and show the accuracy of the i210 based clock in a lightly more complex topology. All of the measurements use my implementation of the master clock (with the Garmin

GPS discipline) as the master of the synchronization system, but on the slaves I measured the offset with respect to the Meinberg's reference 1PPS signal.

The Figure 4.14. shows the measurement records of a system where the switch does not support any kind of PTP functions but both of the nodes have hardware timestamping functionality. As one can see, the parameters of the offset are within the desired specifications stated in the thesis proposal. The maximal peak on the histogram with respect to the reference PPS signal implies that the clock of the slave system gains time. There may be two causes of it, first the switch may introduce error in the delay computation, second the GPS signal of the custom receiver design is overcompensated. In some systems this kind of acausality is forbidden, but it can be circumvented with a different GPS delay value (e.g. the smallest value on Figure 4.9.) When we replace the switch with a transparent clock (in reality I switched the TC mode on the switch), the measurement results (Figure 4.15.) show that the acausality is reduced, and the standard deviation decreased, which was the expected behavior. In both cases I generated no traffic on the switch to collect some baseline information about the synchronization system.

In the next two cases I used the traffic generator of the measurement system. I used the `iperf` utility to generate artifical traffic on the equipment. In previous papers [21] I presented the effect of the client/server roles on the synchronization accuracy, therefore, I used the setup where the most obvious performance hit occurs. On the PTP slave clock I ran the iperf in the master role (traffic discarder – heavy load on the Rx traffic), and the traffic generator was always the client (traffic generator). The master clock did not take part of the traffic generation. I used TCP traffic in the tests because it automatically uses all of the safely available bandwith. The heavy load on receive side traffic (observed from the slave) means that in the switch the sync and delay response packets of the master are queued stochastically.

The uneven reception of the sync and delay response packets highly degrades the traffic as we can observe on Figure 4.16. The PI servo of the ptp4l cannot lock onto the intervals calculated from the message transmission/reception, because the path delay estimation is periodically fluctuates. If the residence time of the packets in the switch is measured (and written out into the packets of course), the non–deterministic jitter of the sync & delay response packet is dropped from the equation, therefore it provides a higher quality estimator of the offsets and delays. The Figure 4.17. depicts the effects of the residence time correction on the accuracy. The mean and standard deviance of the error signal equals to the scenarios without any generated traffic. The wander of the signal is the effect of the daily wander of the Garmin receiver's PPS output, compared to the chosen absolute reference (Fig. 4.9.).



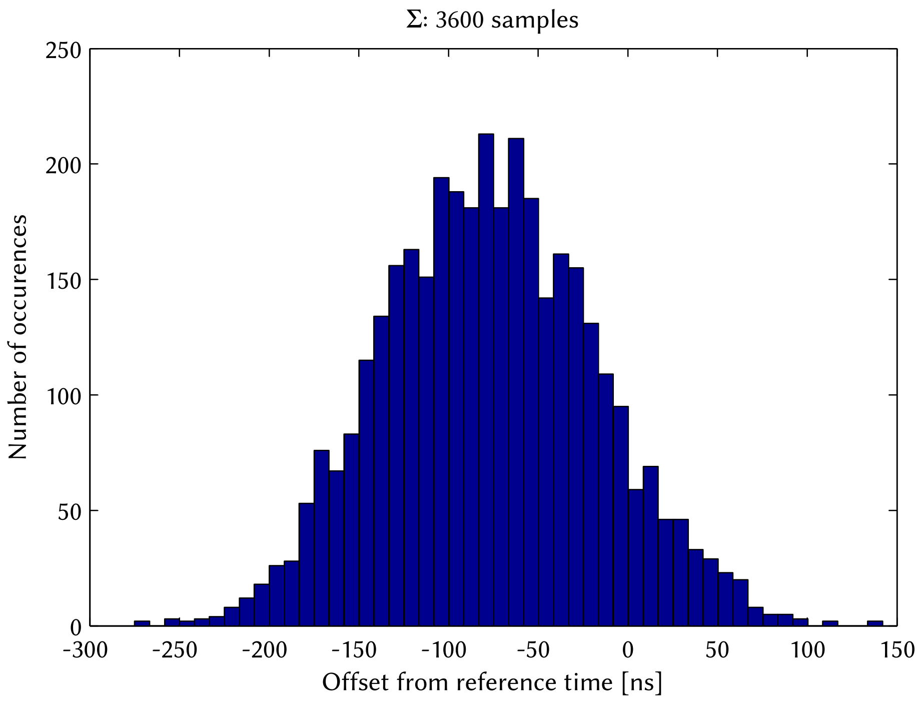
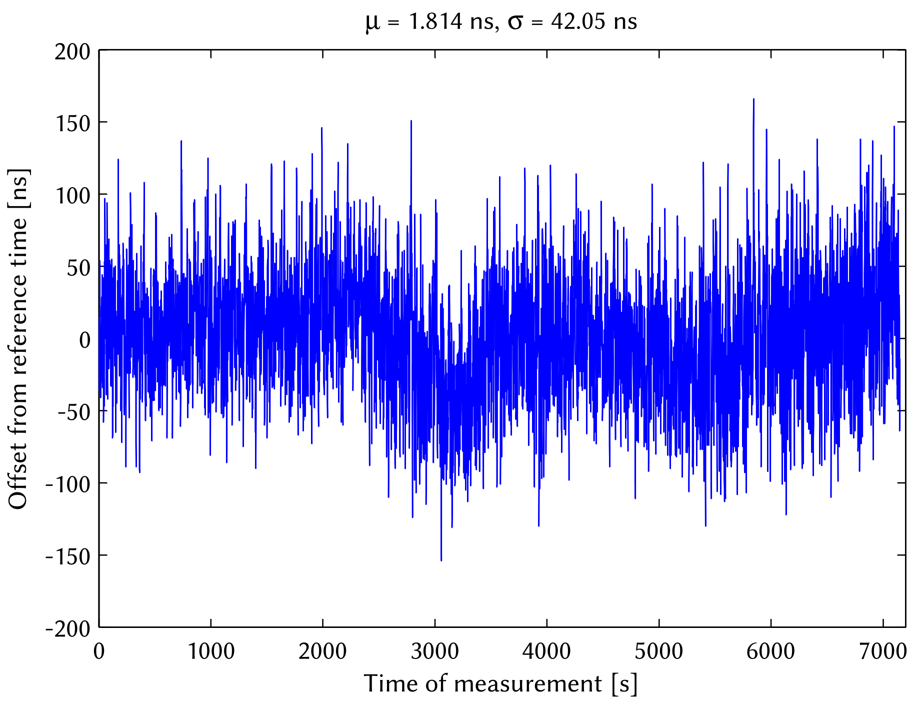


Figure 4.14: The offset from the **absolute reference** with a switch without load (time series & histogram)



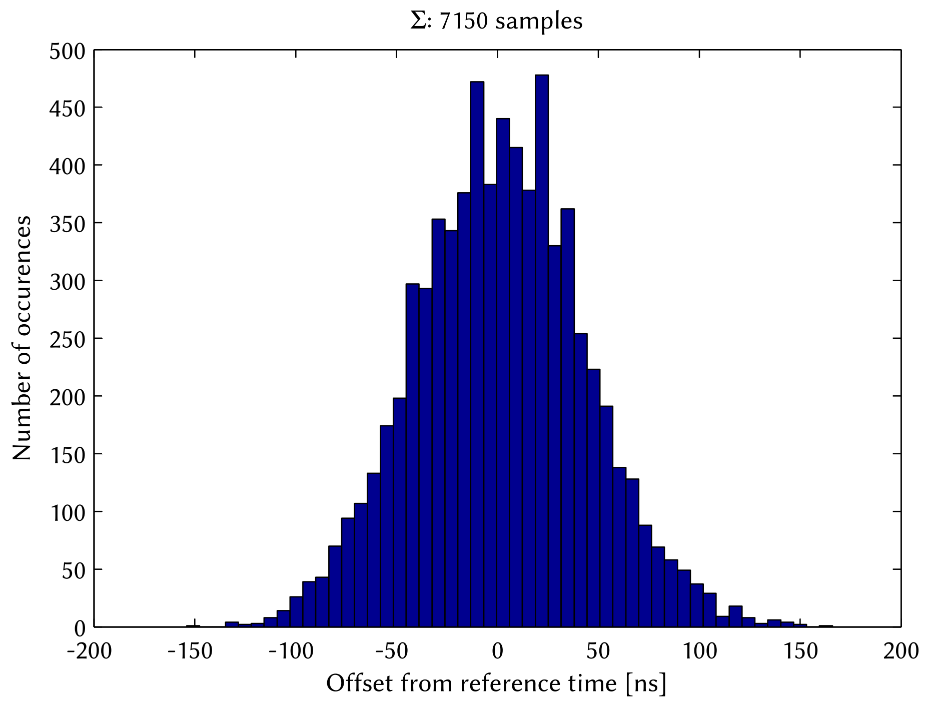
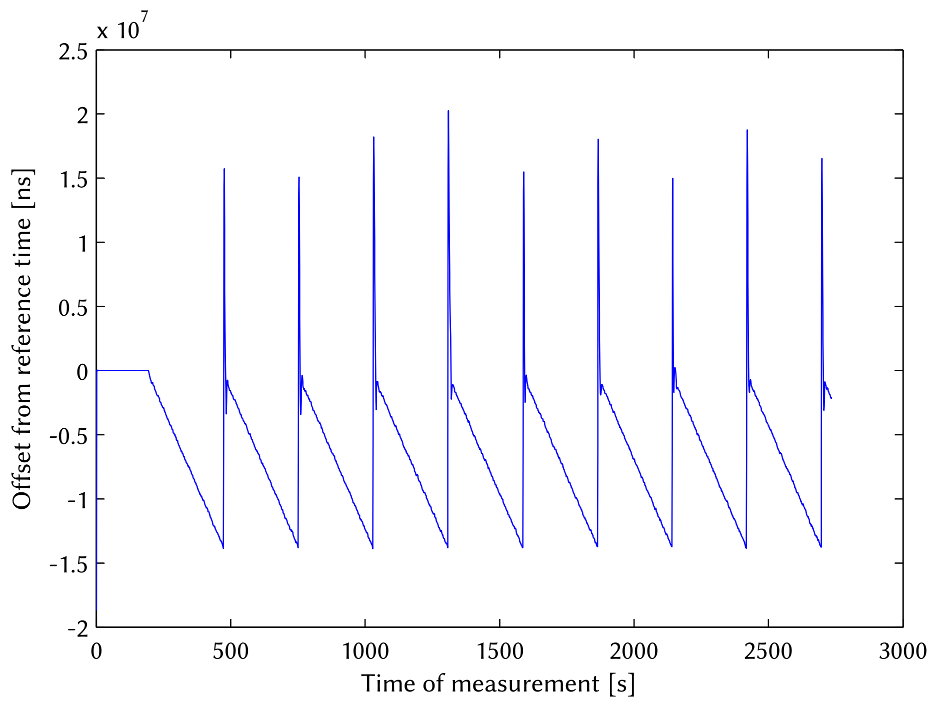


Figure 4.15: The offset from the **absolute reference** with a transparent clock without load (time series & histogram)



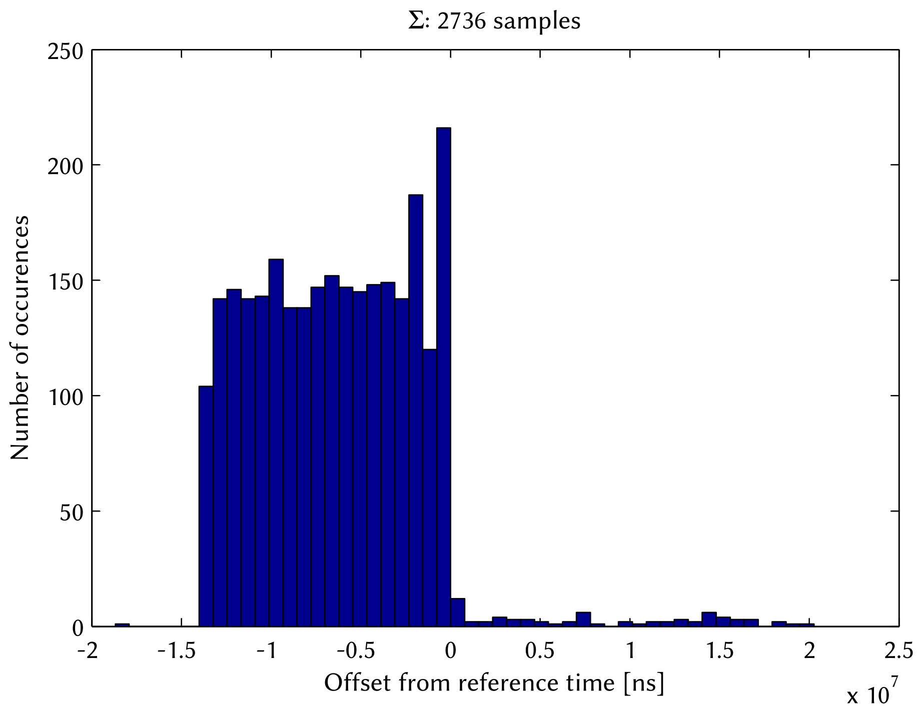
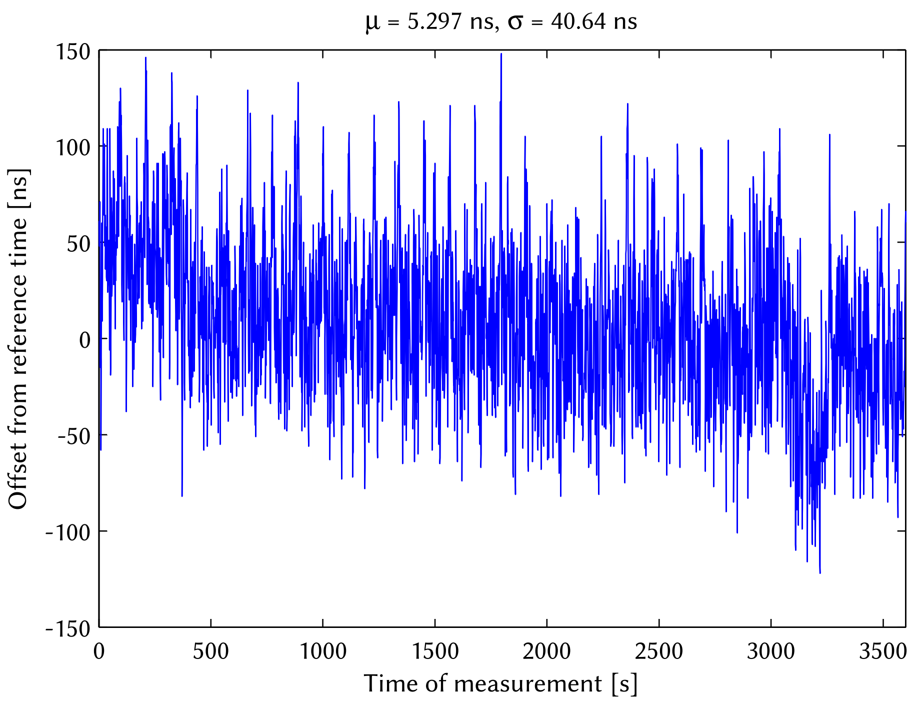


Figure 4.16: The offset from the **absolute reference** with a switch with load (time series & histogram)



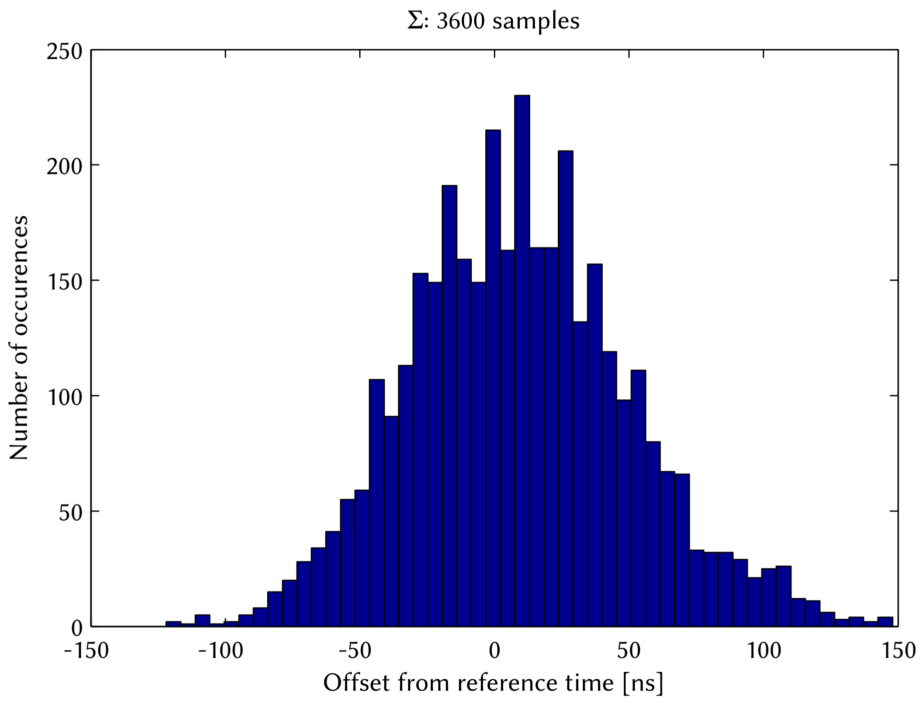


Figure 4.17: The offset from the **absolute reference** with a transparent clock with load (time series & histogram)

5 | Industrial applications and future directions

Sub–microsecond level of accuracy in timing is required in many fields of distributed systems. I've collected some interesting areas where the good quality time–base is mandatory for the system's proper functionality. In the descriptions of them I also emphasise the related research ways of the presented system.

5.1 Common applications of packet based time synchronization

5.1.1 Usage in distributed measurement systems

To ensure the reconstruction of the order of the measurements in a distributed system, the local clock of the measurement nodes have to closely follow the flow of an arbitrary reference time. In non–wireless applications the distribution of the local clock mostly done on the shared communication media. Most of the distributed measurement standards require the implementation of a subset of the IEEE 1588:2008 protocol to attain the required accuracy.

One example of the high–accuracy distributed measurements is the synchrophasor reconstruction in power substation automation applications [22]. The measured current – voltage pairs are time stamped in the sensors and sent to the data processing nodes. The *IEEE C37.118.1-2011 Standard for Synchrophasor Measurements for Power Systems* states the following: "A phase error of 0.57 degrees (0.01 radian) will by itself cause 1% Total Vector Error (TVE) ... This corresponds to a time error of ±26 μs for a 60 Hz system and ±31 μs for a 50 Hz system". It also states that "A time source that reliably provides time, frequency, and frequency stability at least 10 times better than these values corresponding to 1% TVE is highly recommended.". The required accuracy of the timestamps are 2.6 μs in 60 Hz and 3.1 μs in 50 Hz systems. Without proper synchronization, the measured data will be reconstructed in a false order, which makes the evaluation of the events impossible. The sub–µs accuracy helps the operators to make faster and better decisions, compared to the older methods; the possibility to collect tens to thousaunds of sampling points per second provides superior feedback to the controlling personnel.

There are commercially available products, which provide the time–base in these applications, but the system presented in this paper is also capable act as grandmaster in these systems. There are few issues unimplemented right now (such as SNMP capabilites), and there are several requirements which are untested in the current configuration (Peer to Peer TC, VLAN tagging). Some of the unresolved issues are in the scope of further work.

5.1.2 Audio-video bridging

Nowadays the majority of the home entertainment systems use digital interconnections (DVI, S/PDIF, HDMI) to connect the sources (CD, DVD, BluRay players) to the endpoints (TV, Amplifier). By the usage of these interconnections, the integration capability of the devices into home automation systems are poor. In professional A/V systems the usage of the master clock or genlock signals is common, but maintaining the synchronized state of the system is an enormous effort in the ever–changing connection standards. The usage of the ubiquitous Ethernet as the communication medium helps the integration, but poses problems to maintain the quality of service (lack of A/V sync, dropouts). The industry proposed the *802.1 AVB standard family* to resolve these problems. The *IEEE 802.1AS Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks* standard defines a subprofile of the PTP standard covered in this document. The standard set also proposes methods to solve the traffic shaping, stream integrity and bandwidth reservation problems, but it's not in the scope of this document [23].

The main goal in the converged audio–visual systems to provide several streams simulteanously, coexisting with the other traffic on the communication medium. With the usage of time synchronization the participating nodes can recover the reference clock from an arbitrary stream. Every node in the AVB domain synchronizes its clock to the elected grandmaster, and the stream source time stamps the data with respect its internal clock. The playing end–nodes regenerates the media clock with the help of the attached time stamps and its synchronized onboard clock. The accuracy requirements of an AV stream is in the order of milliseconds (lip–syncing); however, maintaining the media clock requires sub–µs accuracy synchronization to implement a good quality distributed PLL.

The presented system introduces some solutions to adapt AVB services, for example disciplining a DAC clock, but it's not a fully–fledged solution. The AVB software stack is not covered in this thesis, but there is at least one freely usable AVB framework under development [24]. The implementation of distributed PLL is also available with the presented system, the slave PTP nodes can emit arbitrary frequency output signals derived from the master's 1PPS input.

5.1.3 Financial systems

In the financial sector the precise and accurate the timing of events is invaluable both for the trading parties and the controlling bodies of the market. The accurately time stamped data of transactions serves as proof of legibility and as an input for data–mining algorithms to determine the optimal trading strategy. The trading software gain their notion of time in most likely from the Linux kernel; therefore a low–latency access to the accurate time is a must–have. The accurate time base also needed for the measurement of network delays, which is a tool to optimize and reduce the time required to access the financial information.

The majority of these system are running on a virtualization platform. The distribution of the accurate time into the VMs is a challenging problem, which requires further investigation. The technologies like *I/O virtualization* (IOMMU) or sharing a single resource between multiple VM instances (*SR–IOV*) — like the timing resources of an adapter — is an interesting problem to discover.

In the presented system, the `phc2sys` tool is capable to discipline the kernel clock to the PHC, but currently independent measurement methods are unavailable in my environment (e.g. a CPU tracer); therefore the sub–µs accuracy is not proved with respect to the kernel timer. The modern Ethernet adapters support some sort of DMA mapped into the CPU cache address space (DCA), which may be the core of serving accurate time to the legacy applications. There are commercial solutions which are capable to achieve sub–µs accuracy in kernel–time synchronization [25].

5.2 Further directions & conclusion

The current system has the a few unsolved issues, which are required for the adoption in production environment. The automatic distribution of the change in the external reference is unavailable, the interoperation between various vendors is currently untested. The serial output of the system is unimplemented yet, and the I/O ports to external equipment should be upgraded to make it usable in production systems. With the usage of coaxial terminals, the EMC of the final product will be greatly improved.

The initial measurements showed that this concept can be used in the majority of measurement, financial and entertainment systems, as it's cheap, reliable and accurate solution for the distribution of timing information. The underlying software architecture has an active development community, and most of the additional components are versatile enough to use it in a alternatively configured system. The accuracy of the synchronization in the hardware is proved with independent reference equipment, but the measurement of the software side is in the scope of further analysis.

Acknowledgement

First of all I would like to thank the help of my supervisors, *Tamás Kovácsházy* and *Zoltán Fodor*. They introduced me to the field of engineering in practice and provided me the necessary knowledge and equipment since 2010 to carry on the writing of this thesis.

On the second I want to thank the support of my family, especially of my parents', and my late grandmother's support throughout the years of studying, and the help of my brother Barnabás who helped me correct the language used in this document.

List of Figures

References

[1] G. Neville-Neil, "PTPd webpage." http://ptpd.sf.net/ (Mar. 2013).

[2] R. Cochran and C. Marinescu, "Design and implementation of a PTP clock infrastructure for the Linux kernel," in *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2010 International IEEE Symposium on*, pp. 116–121, 2010.

[3] R. Cochran, "The Linux PTP Project Homepage." http://linuxptp.sf.net/ (Apr. 2013).

[4] U.S. Navy, "Systems of Time." http://www.usno.navy.mil/USNO/time/master-clock/systems-of-time (Apr. 2013).

[5] T. C. on Sensor Technology (TC-9), *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. IEEE Instrumentation and Measurement Society, 2008.

[6] G. M. Garner, "Description of Use of IEEE 1588 Followup Peer-to-Peer Transparent Clock in A/V Bridging Networks," tech. rep., Samsung, 2006.

[7] J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers, Third Edition*, ch. Time, Delays, and Deferred Work, pp. 183 – 212. O'Reilly, 2005. available at: http://www.iitg.ernet.in/asahu/cs421/books/LDD3e.pdf.

[8] man-pages project, "Time (7) man pages." Available at: `man -s 7 time`.

[9] Google Official Blog, "Time, technology and leaping seconds." http://googleblog.blogspot.hu/2011/09/time-technology-and-leaping-seconds.html (May 2013).

[10] G. Blewitt, *Geodetic Applications of GPS*, ch. Basics of the GPS Technique: Observation Equations, pp. 9–55. Swedish Land Survey, 1997.

[11] GCAPTAIN STAFF, "SARSAT Satellite Tracking with Google Earth." http://gcaptain.com/find-satellites-google-earth/ (May 2013).

[12] D. Goodin, "How to bring down mission-critical GPS networks with $2,500." Available at: `http://arstechnica.com/security/2012/12/how-to-bring-down-mission-critical-gps-networks-with-2500/` (March 2012).

[13] ARINC Engineering Services, LLC, "Navstar GPS Space Segment/Navigation User Interfaces," Available at: `http://www.losangeles.af.mil/shared/media/document/AFD-070803-059.pdf` (Jan. 2013).

[14] U.S. Navy, "USNO NAVSTAR Global Positioning System Information." `http://www.usno.navy.mil/USNO/time/gps/gps-info` (May 2013).

[15] Intel, "Hardware-Assisted IEEE 1588 implementation in the Intel® IXP46X Product Line," white paper, 2005.

[16] Garmin International Inc., *GPS 18x Technical Specifications*, January 2008.

[17] LAN Access Division, *Intel® Ethernet Controller i210 Datasheet.* Intel Corp., 2.1 ed., November 2012.

[18] P. van der Does, "Compile the mainline Linux kernel for Ubuntu." Available at: `http://blog.avirtualhome.com/compile-mainline-kernel-ubuntu/` (Jan. 2013), 2012.

[19] LAN Access Division, *Intel® Ethernet Controller i350 Datasheet.* Intel, 2.05 ed., December 2011.

[20] Meinberg, *Manual of the LANTIME M600 GPS PTPv2 Network Time Server and PTPv2 Grandmaster Clock.*

[21] T. Kovácsházy and B. Ferencz, "Nagy pontosságú IEEE 1588 (PTP) óraszinkronizációs protokoll implementációk fejlesztése és alkalmazása.," in *Networkshop 2013, Sopron*, 2013.

[22] Symmetricom, "Profile for Use of IEEE 1588™ Precision Time Protocol in Power System Applications." Available at: `http://www.symmetricom.com/media/files/resources/webcasts/PTPPowerProfilePresentation_Webinar.pdf` (May 2013).

[23] R. Boatright, "Understanding ieee's new audio video bridging standards." Available at: `http://www.embedded.com/design/connectivity/4008284/Understanding-IEEE-s-new-audio-video-bridging-standards` (Dec. 2012).

[24] Intel, "Open AVB webpage." `https://github.com/intel-ethernet/Open-AVB` (Apr 2013).

[25] Symmetricom, “Delivering Sub-Microsecond Accurate Time to Linux Applications Around the World.” Available at: `http://www.symmetricom.com/media/files/resources/webcasts/Webinar-Low-Latency-Dec-04-2012.pdf` (May 2013).

Appendix

A.1 Obtaining the software and source code

In this section I provide the availability all of the software and source codes, which are used in this paper.

A.1.1 igb–pps & utilites

The modified igb driver and the corresponding utilites are maintained by the author (Bálint Ferencz, <ferencz.balint@sch.bme.hu>). It's hosted on the Atlassian Bitbucket version control system. One can download it by issuing the following command:

```
git clone https://bitbucket.org/fernya/igb-pps.git igb-pps
```

The project is currently in embryonic state so I haven't provided any tagged tarball for distribution.

A.1.2 Linuxptp

The linuxptp software is maintained by Richard Cochran, and it is hosted as a Sourceforge project [3]. Since the introduction of the version 1.0, the software gets released in tarball format. For those who are interested in the nightly commits, the (read–only) git repository may be cloned with issuing the following command:

```
git clone git://git.code.sf.net/p/linuxptp/code linuxptp
```

A.1.3 Garmin setup utility

To configure the GPS receiver I have used the Windows based configuration tool. The usage of the utility is straightforward, after a successful connection, the properties of the receiver are editable in a dialog box. After saving, the new settings can be applied to the receiver with serial uploading. The majority of the settings are effective after upload, but for the measurement output one needs to repower the receiver. The program is available from the following place:

```
http://www8.garmin.com/support/agree.jsp?id=4071
```