TU WIEN Informatics

Drahtlose Synchronisierung von tragbaren Sensorknoten

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Matthias Prader, BSc
Matrikelnummer 01625755

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Inf. Dr.sc.techn. Florian Michahelles
Mitwirkung: Univ.Ass. Dr.-Ing. Florian Wolling, MSc

Wien, 25. März 2025

Matthias Prader	Florian Michahelles

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Wireless Synchronization of Wearable Sensor Nodes

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Matthias Prader, BSc
Registration Number 01625755

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Inf. Dr.sc.techn. Florian Michahelles
Assistance: Univ.Ass. Dr.-Ing. Florian Wolling, MSc

Vienna, March 25, 2025

Matthias Prader	Florian Michahelles

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Matthias Prader, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, habe ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 25. März 2025

Matthias Prader

Danksagung

Allen voran möchte ich meinen Eltern danken! Danke für euer Vertrauen in mich und eure bedingungslose Unterstützung!

Ein großes Dankeschön gilt auch Dr.-Ing. Florian Wolling, für die Betreuung dieser Arbeit, die dabei aufgebrachte Geduld und eingebrachte Expertise!

Danken möchte ich auch meiner Partnerin und all den lieben Menschen in meinem Umfeld, denen mein Erfolg am Herzen liegt und die mich sowohl bei dieser Arbeit als auch darüber hinaus stets unterstützen und begleiten.

Kurzfassung

Tragbare Sensorknoten sind kompakte, mit Sensoren ausgestattete Recheneinheiten, die am Körper getragen werden können. Diese Geräte können in verschiedenen Kombinationen für die Datenerfassung in realen Szenarien verwendet werden und ermöglichen somit Forschung außerhalb von Laborbedingungen. So können beispielsweise mehrere Sensoren eingesetzt werden, um Aktivitäten in Gruppentreffen für sozialwissenschaftliche Studien zu tracken oder physiologische Parameter und Bewegungsparameter von Sporttreibenden während des Sports gesammelt werden, um ihr Training zu optimieren und Verletzungen vorzubeugen. Wenn mehrere Sensoren Daten erfassen, müssen diese in der Regel zusammengeführt werden, um sinnvolle Erkenntnisse zu gewinnen. Ereignisbasierte Datenzusammenführung stützt sich auf Muster in den Sensordaten, funktioniert daher nur mit kompatiblen Sensorkombinationen und wird durch Messungenauigkeiten beeinträchtigt. Zeitstempelbasierte Datenkombination hingegen ist unabhängig von den Sensoren und deren Messungen, benötigt jedoch eine genaue Zeitsynchronisation. Lösungen für genaue Zeitsynchronisation von tragbaren Sensoren basieren oft auf bestimmter Hardware und sind daher nicht portabel. Das Simple Network Time Protocol (SNTP), eine Vereinfachung des Network Time Protocol (NTP), stellt eine softwarebasierte, portable Lösung dar, ist jedoch aufgrund von asymmetrischen Paketverzögerungen ungenau.
Diese Arbeit untersucht portable, auf Software der Anwendungsschicht basierte, drahtlose Zeitsynchronisation von tragbaren Sensorknoten. Um verschiedene Zeitsynchronisierungsansätze unter kontrollierten Bedingungen und mit simulierten Umwelteinflüssen zu untersuchen, wird ein Prüfstand entwickelt. Es werden zwei Ansätze analysiert und miteinander verglichen. 1) SNTP, die Standard-Zeitsynchronisierungsmethode des beliebten ESP32 Mikrocontrollers; 2) RAMSES - Repeated Averaging of Multiple SNTP Executions for Synchronization - eine neue Methode zur Zeitsynchronisierung, die in dieser Arbeit vorgeschlagen wird. RAMSES basiert auf SNTP und erreicht verbesserte Ergebnisse durch Ausfiltern von Ausreißern und statistische Auswertung mehrfacher Ausführungen des On-Wire Protokolls. Untersuchungsergebnisse zeigen, dass RAMSES gegenüber SNTP eine signifikante Reduktion der Standardabweichung der relativen Zeitdifferenzen zwischen synchronisierten Geräten von bis zu 78% ($p<0.001$) erreicht. Aufgrund der erheblich verbesserten Stabilität der Zeitsynchronisierung ist RAMSES für Anwendungen, die eine höhere Genauigkeit erfordern, eine geeignete Alternative zu SNTP.

Abstract

Wearable sensor nodes are compact computing units equipped with sensors that can be worn on the body. These devices can be used in various combinations to collect data in real-life scenarios, enabling research beyond laboratory settings. For example, multiple sensors could be utilized to track activities at group gatherings for social science studies or collect physiological and motion parameters of athletes during exercise to optimize training and prevent injuries.

The data collected by multiple sensors must usually be combined to derive meaningful insights. Event-based data combination relies on patterns in the sensor readings, is thus constrained to be used with compatible sensor combinations, and is affected by measurement inaccuracies. Timestamp-based data combination on the other hand is independent of sensors and their readings but requires accurate time synchronization. Solutions for accurate time synchronization on wearable sensors often rely on certain hardware and thus lack portability. The Simple Network Time Protocol (SNTP), a simplification of the Network Time Protocol (NTP), offers a software-based, portable alternative but suffers from inaccuracies due to asymmetric packet delays.

This thesis explores portable, application-layer software-based wireless time synchronization for wearable sensor nodes. A test bench is developed to study different time synchronization approaches under controlled conditions with simulated environmental influences. Two approaches are analyzed and compared: 1) SNTP, the default time synchronization method of the popular ESP32 microcontroller; 2) RAMSES - Repeated Averaging of Multiple SNTP Executions for Synchronization - a new method for time synchronization proposed in this work. RAMSES is based on SNTP and achieves improved results through outlier filtering and statistical evaluation of multiple on-wire protocol executions. Experimental results show that RAMSES significantly reduces the standard deviation of th e relative time differences between synchronized devices by up to 78% ($p < 0.001$) compared to SNTP. Due to the substantially improved stability of time synchronization, RAMSES is a viable alternative to SNTP for applications that require increased accuracy.

Contents

CHAPTER 1

Introduction

The technological advances in recent years have made computer hardware both smaller and more affordable. As a result, computing devices have become pervasive as they found their way into our everyday lives. Smartphones, smart rings and fitness trackers are prominent examples of this development. These innovations not only bring various benefits for businesses and consumers, but they also open up attractive new opportunities for scientific research. Wearable sensors have been used to study human behavior in more natural environments than previously possible [RTH20] and they have enabled research in the medical field such as to detect Parkinson's disease several years in advance [SPHS23] and to better triage and handle hospitalized patients with dengue. [MDC+24]

Because technology has become so economical, small, and powerful, it is now feasible to use multiple independent wireless sensors to collect data in real-world scenarios. Things that could previously only be analyzed in laboratories or with restrictive cabling are therefore easier to study, possibly even in their natural environment. Since the multiple sensors typically collect data about related or the same research subjects, that data will eventually be collected and combined for processing. When doing so, it is often required to know for different data points whether they were registered simultaneously [GED+22], which one happened before, how much time elapsed between the registration of the data points, etc.

A possible solution is the synchronization based on events [BAL09]. During the data collection, actions that produce a certain signature in the sensor readings are performed. Such an action could be shaking the sensors together in case Inertial Measurement Units (IMUs) are used [SGO+23]. This approach requires the use of sensors that are compatible with each other in the sense that there are actions that produce characteristic sensor readings. There could be sensor combinations for which there are no such actions, or the effects of actions on certain sensors might be the subject of research. Additionally, the accuracy achieved by synchronizing based on events has been reported to be in the tens of milliseconds. [AAR+20][WSL+19]

In other domains, for example in our social life, the time is used to determine the sequence of events, their duration, and the distance between them. Directly assigning a timestamp to each data point during data acquisition would greatly facilitate the combination and further processing of the data from different sensors. However, for time to be a meaningful value in a system, all actors within that system must interpret time in the same way. In other words, a common time needs to be established across all clocks in the system that assigns timestamps. An important parameter of time synchronization is the achieved accuracy because it determines the maximum time precision that can be used in a system. Because the components in wearables that are responsible for timekeeping typically exhibit clock drift (section 2.2), it is required to repeatedly synchronize the clocks. From a user's perspective, wireless communication is a convenient way to do so. However, wireless communication is known to be much less reliable than wired connections. This is expressed by data being lost on the way from sender to receiver and varying transmission times due to changing environmental conditions. [HHN+07]
In the Literature, most of the proposed solutions for wireless clock synchronization on wearables operate rather close to the hardware and partly use some form of specialized hardware. This includes Media Access Control (MAC) layer timestamping, dedicated circuits to detect synchronization signals, and leveraging hardware side channels. [HIK+22][YTLH19][HGBR21]
While these approaches achieve good results, they lack portability because they rely on specific hardware.
A very prominent and portable solution for time synchronization on network devices is the Network Time Protocol (NTP). The protocol was designed to synchronize time over the internet and without time-synchronization-specific hardware requirements. With the Simple Network Time Protocol (SNTP) there is also a simpler and more lightweight version of NTP. While these are desirable properties, the accuracy, especially for SNTP, depends a lot on the network conditions. In particular, asymmetric network packet transmission delays can cause errors of tens to hundreds of milliseconds. While this might be fine for some applications, it most probably will be too much for the task of aligning data collected by different sensors based on timestamps. A typical scenario in which multiple sensors collect data that needs to be combined in the aftermath for further processing is activity recognition. For such tasks, sampling rates of about 50 Hz to 100 Hz have shown to be a good fit. [KHMP16]

If the same clock source were used to timestamp sensor readings, they would be trivially ordered. In cases where multiple clock sources are used, they need to be synchronized. Note, however, that it is not need to synchronize them with a global time source. For the described purpose, it is sufficient that all time sources are synchronized to the same main clock. Since at 100 Hz, a value is expected every 10 ms, the goal is to achieve a synchronization accuracy of 5ms. This way, two clock sources in the system are at most 10ms apart.

This thesis is structured as follows: background information on wearables and time synchronization is provided in chapter 2 followed by a discussion of related work in chapter 3.

To establish a synchronization accuracy baseline, the default way of synchronizing time on a popular Microcontroller is evaluated. Based on these findings, an improved time synchronization approach is proposed. The methodology is presented in chapter 4, while the Implementation details of the proposed synchronization approach and the setup used to assess synchronization performance are outlined in chapter 5. The achieved results are evaluated in chapter 6 and discussed in chapter 7 before the thesis is concluded in chapter 8.

CHAPTER 2

Background

2.1 Wearable Computing

The term wearable computing refers to the research, development, and operation of computer devices that are worn on the user's body. The idea behind wearable computing is that the user is constantly accompanied by a computing device that is operating all the time to be of service to its wearer. The probably most prominent example of wearable computing devices today is the smartwatch.
Steve Mann, considered the "father of wearable computing"[1], defined six properties for wearable computing devices [Man01]:

1. *Unmonopolizing of the User's Attention:* Users can concentrate on other things while wearable computing runs on the side.
2. *Unrestrictive to the User:* Wearable computing should not restrict the user's actions. For example, it can be used in lying down as well as while jogging.
3. *Observable by the User:* If wanted by the user, wearable computing can get the user's continuous attention via an always perceptible output interface.
4. *Controllable by the User:* The user of a wearable computing device must at all times be able to take control of the device.
5. *Attentive to the Environment:* Wearable computing devices should have the ability to perceive their surroundings via multiple sensors and thus enable the user to get an increased perception of the environment.
6. *Communicative to Others:* Wearable computing devices should be able to perform direct communication with others or assist users in expressing themselves.

[1] https://discover.research.utoronto.ca/1297-steve-mann

A Core component of wearable computing devices is a computing unit with various inputs and outputs and wireless communication [ZZZ+22].
Since wearable computing devices, often called wearables, are worn by their users and move with them they cannot be connected to mains power or wired networks. They are typically battery powered and thus power consumption is a big concern when such devices are designed. This results in wearables being constrained in processing power and memory to not consume more power than needed. Due to the limitations in processing power, memory, and power supply, wearables are **resource constrained devices**
Because of the battery-based power supply and wearables moving together with their user, wearables are constantly subject to changes in parameters like supply voltage or temperature. Hence, wearables are operating in **dynamic environments**.
Together with wireless communication, dynamic environments lead to changes in interference, which causes changing transmission times and variable packet loss [HHN+07].

2.2 Time Discrepancies

Most clocks used today rely on some form of oscillator, mostly quartz crystal oscillators, to measure the time. Oscillators produce a constant, periodic signal. The basic idea behind oscillator-based clocks is to count distinctive markers of that signal and, by knowing the frequency of the signal, keep track of the time. Thus, the oscillator makes the clock tick. [TAA19]
Such clocks can be used to assign numbers to observed events within a system, where the numbers represent the chronological order in which the events were observed. These clocks are called logical clocks. [Lam78]
The number a logical clock assigns depends on when the clock was first started. So it is possible to have two distinct systems, each one with its own clock that both assign different values to the same observed event. In this case, one clock has a **clock offset** relative to the other clock. To counter this, there is a process to establish a common time across clocks called clock synchronization. Ideally, after the synchronization the clock offset is zero and every clock advances time independently. For the clocks to remain synchronized, they need to "tick" at precisely the same time at the same frequency. In practice, oscillators are subject to variations in their frequency [TAA19]. This happens because of changes in temperature, supply voltage, oscillator age, etc. The frequency difference between two clocks at a point in time is called **clock skew**. The variation of the clock skew over time is called **clock drift**. [RFC2330][TAA19]

Clock Attribute	Description
offset	Time difference between two clocks.
skew	"Tick Frequency" difference between two clocks at a point in time.
drift	Skew variation over time.

Oftentimes, the clock skew is assumed to be constant over some time and consequently,

skew and drift are used interchangeably in literature [TAA19].

Xu et al. classified time discrepancies in the specific context of combining data collected from multiple wearables. In their work, they report four different timestamp inconsistencies. This is when different devices collect data at the same time but in the end, different timestamps are assigned to the data. [XGMW17]
With *timestamp scale inconsistency* in the system, devices use different time scales, resulting in inconsistent timestamps. When device A uses time on the seconds scale, device B uses time on the milliseconds scale, and both timestamp the same event, the timestamps are inconsistent because the timestamp of device B is more accurate. *Timestamp frequency inconsistency* is the type of inconsistency when devices use different frequencies to gather and timestamp data. Because of this frequency difference, it is difficult to get a data point from both devices at the same time. Device clock drifts cause *timestamp drift*. There, the timestamp pattern of data collected at regular intervals does not reflect the regularity but the timestamps deviate slightly from the actual intervals. *Timestamp absence* is caused by the lack of device time synchronization and the device assigning timestamps relative to its starting time. Such timestamps cannot be used to combine data from different devices. [XGMW17]

2.3 Time Synchronization

Time synchronization is the process of adjusting the time of different, independent clocks to ensure that all clocks involved in the synchronization process indicate the same time. Time synchronization shapes our everyday lives because a lot that happens is tied to specific times. For most social requirements, synchronization to seconds should be more than enough. When contexts get more technical, however, the required time accuracy can be much smaller.
The scope of time consensus also depends on contextual factors. In group meetings, for example, only the participants need to share a time consensus. This consensus does not have to match the time consensus of outsiders, but due to the impracticality of multiple time consensuses in everyday life, orientation on a global consensus is favorable. In technical systems, on the other hand, it might be advantageous to not match the global time consensus but to come up with one that is just shared by the participants of the system to reduce complexity.

Network Time Protocol

The Network Time Protocol (NTP) is used to synchronize clocks in a network. It was first introduced in 1985 [RFC958]. The protocol builds on top of the User Datagram Protocol (UDP) and is a pure software application layer protocol (section 2.4). NTP comes with a set of algorithms to determine the offset of the client's clock to the Coordinated Universal Time (UTC) provided by NTP servers. Multiple servers are used to determine this offset. Servers are organized in a hierarchical system. The servers on top of the hierarchy (named stratum 1) get their time directly from high-precision time sources like GPS or

atomic clocks. Servers lower in the hierarchy (stratum 2, stratum 3, etc.) get their time by synchronizing with multiple servers one step higher in the hierarchy (lower stratum number).
Querying multiple servers and running algorithm suites to eliminate possible errors stemming from network packet transportation issues is good for accuracy, however, it might not be suitable or desirable for resource-constrained devices or applications with modest time requirements and thus time accuracy can be exchanged for less complexity. In 1992 the Simple Network Time Protocol (**SNTP**) was introduced [RFC1361]. In contrast to NTP, SNTP only uses one server to determine the client's time offset. Thus, a lot of algorithms are omitted, which makes the protocol simpler, yet it is still compatible with existing NTP servers.

On-Wire Protocol

A cornerstone of (S)NTP is the on-wire protocol. It defines what data needs to be exchanged in what form between the NTP client and NTP server to determine the client's time offset. The basic idea behind the protocol is illustrated in Figure 2.1. The client starts by sending its time to the server (T1). The server records T2, the time at which the message from the client is received, as well as T3, the time at which the server answers the client message. The client receives the server response at time T4. T1 and T4 are recorded with the client's clock, while T2 and T3 are recorded with the server's clock. The message that the client receives as an answer from the server contains T1, T2, and T3.

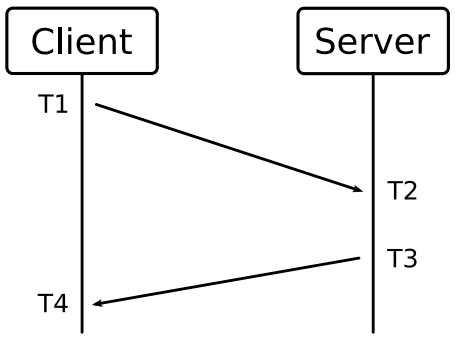


Figure 2.1: On-Wire protocol illustration.

With this information, the client can determine its own offset with respect to the server by applying Equation 2.1 and adjust the local clock accordingly. This offset calculation is used in NTP, and it is encouraged to be used with SNTP too. However, for SNTP it is also allowed to directly update the local time to T3 and call the client time synchronized. [RFC5905]
The client can also determine the round-trip time (Equation 2.2) the packet took. This information is used in some of the algorithms that are part of NTP.

$$client\_offset = \frac{(T2 - T1) + (T3 - T4)}{2} \tag{2.1}$$

$$packet\_round\_trip\_time = (T4 - T1) - (T3 - T2) \tag{2.2}$$

2.4 ISO OSI Model

In the 1970s, the International Organization for Standardization (ISO) created a committee named Open Systems Interconnection (OSI) to establish open standards that enable the intercommunication of computer networks. This work produced a seven-layer model representing an abstract communication architecture known as the ISO OSI Model. Each layer in the model has a specific function and is in principle independent of the other layers. Higher layers provide functionalities utilizing the functions provided by lower layers. Communication only ever takes place between two directly neighboring layers. Table 2.1 lists the seven layers of the ISO OSI model. The first layer, the physical layer, handles the physical data, transmitting and receiving bits. The second layer, the data link layer offers basic communication services including physical device addressing, data frame transmissions, and transmission error detection. The third layer, the network layer, manages network communication with network addressing and packet routing between networks. The fourth layer, the transport layer, offers communication between processes on different systems. The fifth layer, the session layer, offers functionalities for communication sessions. The sixth layer, the presentation layer handles the data formatting and translates between different encodings used by the fifth and the seventh layer. The seventh layer, the application layer, provides network services to the software applications. [Sax14] [Ala14]

7.	Application Layer
6.	Presentation Layer
5.	Session Layer
4.	Transport Layer
3.	Network Layer
2.	Data Link Layer
1.	Physical Layer

Table 2.1: The seven layers of the ISO OSI model.

2.5 Statistical Terms and Concepts

Various statistical terms and concepts are used in this thesis and briefly explained in this section.

Function fitting is the process of determining a function that best approximates a given set of data points. For example when fitting a linear function to a set of data points, the parameters m and b of the function $y = m \times x + b$ can be determined using simple linear regression, a statistical method to find a line that minimizes the sum of distances between the data points and the line.[2]

Coefficient of determination (R^2) is a performance metric for fitted functions that expresses how well the function fits the data points. The typical value range of R^2 is from 0 to 1, where values close to 1 indicate a particularly good fit.[3]

Kernel density estimate (KDE) plots are similar to histograms but rather than showing bins as discrete bars, the KDE plot shows a continuous curve approximating the probability density.[4]

Quartiles are specific percentiles. Percentiles are numbers that divide a set of data points into two parts according to a proportion p. The pth percentile is the value in the dataset for which about p% of the values are smaller or equal than that value and about $(100 - p)$% are greater or equal to that value. A popular percentile is the $50th$ percentile, also known as the median. Quartiles divide the data set into four equal parts. The first quartile is the $25th$ percentile, the second quartile is the $50th$ percentile (median), and the third quartile is the $75th$ percentile. [DKLM05]
The inter-quartile range (IQR) indicates the width of the middle 50% of the dataset and is calculated by subtracting the $25th$ percentile (first quartile) from the $75th$ percentile (third quartile). [DKLM05]

[2] https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/regression-and-correlation/simple-linear-regression.html, accessed on 2025-03-17

[3] https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/regression-and-correlation/coefficient-of-determination-r-squared.html, accessed on 2025-03-23

[4] https://seaborn.pydata.org/generated/seaborn.kdeplot.html accessed on 2025-03-17

CHAPTER 3

Related Work

The Precision Time Protocol (PTP) is a time synchronization protocol defined in the IEEE 1588 standard [45708]. The protocol achieves synchronization accuracies in the nanosecond range in local area networks when special hardware to timestamp network packets is used. When PTP is implemented in software, accuracy in the order of microseconds can be achieved if dedicated hardware and driver support that allow access to the network card's receive/transmit timestamps are available. Pure application-layer software implementations of PTP achieve about the same accuracy as NTP. [CY21]

Mani et al. proposed a packet exchange protocol in 2018 called **SPoT**. The protocol addresses the problem of clock skew in IoT devices and asymmetric network delays when synchronizing their time. According to their evaluation, SPoT is 17x more accurate than SNTP and can maintain a clock accuracy of about 15ms. This is achieved by estimating the clock skew of the device. This clock skew estimation is used to correct the local time readings between synchronizations and to calculate an expected time offset when resynchronizing. When synchronizing, SPoT determines the asymmetry of the network delay using the expected and calculated offset and corrects the calculated offset accordingly. [MDBS18]

Yan et al. proposed **TouchSync**, an approach that achieved wearable clock synchronization with an accuracy below 10 milliseconds using software running at the application layer. Their solution combines an (S)NTP on-wire-like network protocol with an external oscillating signal. This signal is the electromagnetic radiation from powerlines, that is captured by using the human body as an antenna and an ADC to sample the skin's electric potential. Their experiments showed, that the power grid radiation measured at different positions on a person is shifted by less than 1 ms. Also, they showed that this approach also works to synchronize between wearables attached to different persons over a distance of about 10km [YTLH19].

Hofmann et al. presented **X-Sync**, a method of synchronizing time that operates very close to the physical layer. They used a cross-technology communication approach called packet-level modulation, which consists of creating energy bursts by sending data packets of variable length. The sent information is decoded by sampling the received signal strength (RSS) on the client side. Together with layer-2 timestamping and estimating the clock drift using linear regression, they achieve µs-level accuracy in time synchronization of IoT devices [HGBR21].

Wolling et al. [WvLSR21] proposed **PulSync**, a method to align data from multiple wearable devices using the heartbeat of the person wearing the devices. The heartbeat is a persistent signal that varies due to heart rate variability. These variations produce unique patterns that can be used to align data collections. They evaluated their approach using an accessible dataset containing electrocardiogram (ECG) signals collected simultaneously by two devices at a sampling rate of 250 Hz from 25 people and reported an accuracy of -2.856 ± 11.427 ms.

Han et al. [HGTW24] developed open-source hardware for the purpose of synchronizing inertial measurement unit (IMU) sensors. They do this by generating electromagnetic events that are picked up by the magnetometers in the IMUs. The form of the electromagnetic synchronization signal results from previous work of some of the authors [GLD+24]. By creating a square wave signal with varying pulse width they achieve a sub-sample period accuracy with a maximum synchronization error of 8ms at a sample rate of 25 Hz.

Wang et al. [WSL+19] worked at reducing the error that remains after manually synchronizing sensor readings after the data collection process based on context markers in the data that were created by placing multiple wearables with accelerometers on a table and hitting the table with the fist multiple time at the beginning of the data collection process. In their setting, the wearables send their measurements to a receiver, which assigns timestamps to the data points. To improve the results, they first used NTP to determine the receiver's clock drift over multiple hours. By combining the information about the clock drift of the receiver that timestamps the data with the synchronization based on context markers they achieved an error of about 20ms during offline use. The longer the runtime of the data collection processes, the bigger the error reduction in their results: for example after 5 hours, the error without their clock drift reduction was more than 500ms, and for 15 hours more than 1300 ms.

Harada et al. [HIK+22] presented an application layer method to synchronize time using Bluetooth Low Energy. Their method requires a way for the application layer to know when the Bluetooth radio turns active. This can be either by a corresponding functionality of the microcontroller or a dedicated circuit to measure the current consumption to detect the activation of the radio functionality proposed by Rheinländer and Wehn [RW16]. The synchronization works by sending two packets: one with the

current timestamp and one with the timestamp at which the activation of the Bluetooth radio was detected. The receiving device also detects the activation of its Bluetooth radio. With the so obtained timestamps, a time offset can be calculated that is adjusted for delays caused by protocol stacks and the Bluetooth protocol. They evaluated their approach and reported results of an average root mean square error of 20µs.

Luo et al. [LKK+17] experimented with NTP on smartphones and showed that the accuracy of individual NTP synchronization results is subject to variations in the order of 100 ms. They found that combining 10 or more NTP synchronization results brings significant accuracy improvements. For their experiments they used three smartphones, each synchronizing every minute with an NTP server over WiFi. The smartphones were attached to each other and swung like a pendulum. The movement was recorded using the built-in accelerometers. With the accelerometer data they show that when using 10 NTP synchronization results, the average error between two smartphones is 27ms.

CHAPTER 4

Methodology

Wearables are resource-limited devices that are body-worn, move with their wearer, and use wireless communication technologies to exchange information. Both the body and the environment on or in which the wearables are located influence wireless connections. In this thesis, time synchronization possibilities to operate multiple independent wearables in a composite system with a limited spatial spread are investigated. In this setting, it is sufficient to have local time synchronization. Global time synchronization, for example to Coordinated Universal Time (UTC), is not necessary, because when multiple wearables assign timestamps to collected data points for the purpose of using them to later merge the data from the devices, the actual value of the timestamps is completely irrelevant. However, it is crucial, that all wearables assign the same timestamp to an event. To assess the performance of different time synchronization approaches, data must be collected over several hours to days for each approach. Collecting this data with body-worn wearables is impractical due to the long periods and cannot guarantee the needed reproducibility due to the high interaction of real wireless communication with the body and the environment. Because of this, a test bench is going to be built on which the devices will be operated stationary and the effects of body and environment on wireless communication will be simulated in a reproducible fashion. The test bench will be composed of multiple resource-constrained microcontrollers similar to those used in wearables and a time source, with which the microcontrollers will synchronize their time. Within the test bench, multiple measurements will be conducted to record metrics of the time synchronization quality. A potentially suitable metric for this purpose is the relative time difference between the devices. For time synchronization, the devices in the test bench will use some form of network communication. The test bench simulates interference and influencing factors on the network communication to resemble real-world wireless communication of wearables where the signals are affected by the environment and body. The test bench should have the ability to assess the performance of different time synchronization approaches in different environments. Therefore, the time synchro-

nization approach and environment simulation parameters will be input values to the test bench. The output of the test bench will be the collected metrics about the time synchronization quality. Statistics on the test bench outputs will be obtained and used to compare different synchronization approaches.
In this work, the devices will be operated without temperature control in residential space and with constant voltage from a power source with sufficient capacity. For simplicity, it is assumed that temperature and power supply do not influence the clock drift of the devices. All other disturbances and influencing factors are either abstracted to simulation parameters affecting the application layer or neglected.

Based on the described concept, the following research questions are elaborated on in this work:

RQ1 How can time on wearables be synchronized wirelessly?

RQ2 How can time on wearables be synchronized with an accuracy of more than 5ms using an application-layer software approach?

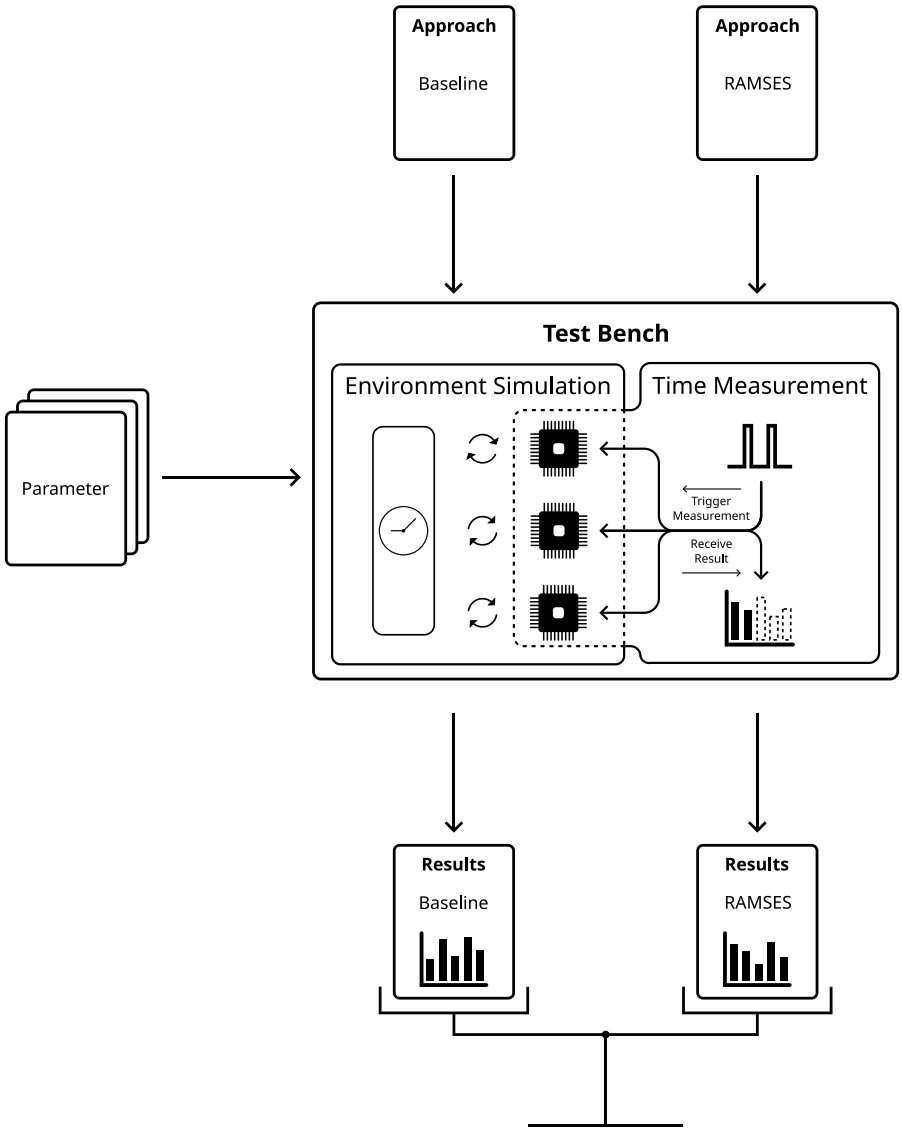


Figure 4.1: In this work, two synchronization approaches are compared by analyzing their performance using a custom-built test bench. The test bench simulates wearable environments based on parameters that can be varied. In this simulated environment, multiple wearable-like devices are operated and synchronized to a time source using the selected approach. Within the test bench, a signal source repeatedly triggers time measurements that are collected and eventually output as the test bench result.

CHAPTER 5

Implementation

For the implementation part of this thesis, a hardware platform to experiment with is needed. The cornerstones of this work are wearable devices capable of wireless communication to synchronize their time. Although when talking about wearables power consumption is an important factor, since building a test bench, it was decided to not place too much importance on this aspect in the selection process for a hardware platform and the ESP32 from Espressif was chosen. The ESP32 is a popular microcontroller in the field of the Internet of Things that comes with WiFi and Bluetooth for wireless communication. This microcontroller is not only used in commercially available products but has also been used in scientific work in the past.
As for wireless communication, WiFi was initially chosen instead of Bluetooth. Using WiFi allows the use of simple networking equipment to set up a network, simplifies the connection of multiple devices, and enables the use of networking tools to introduce noise in the form of delays or packet loss to evaluate different synchronization approaches under different conditions. Finally, no wireless communication at all was used in the test bench but a wired network in which packet loss and delay were simulated. Section 5.1 contains more details about the test bench design, its structure, the challenges and decisions around the software running on the ESP32s as well as the software responsible for collecting the time measurements in the setup. Section 5.2 describes the configuration of the default ESP32 time synchronization capabilities. The proposed synchronization approach to improve results is outlined in section 5.3.

5.1 Test Bench

To evaluate different synchronization approaches, a standardized method for quantifying their quality using systematically gathered data is developed. The most interesting quality aspect is the relative time deviation between synchronized devices. To determine

this quality, the device time of synchronized test devices is repeatedly captured at the same external times and the difference between them is calculated.

To capture the device-time of the synchronized test devices at the same external times a single signal source is deployed and connected to every test device via a wire. Every time the signal source produces a pulse, each device stores the time at which it receives the pulse. This way the device times (possibly one distinct for every test device) are recorded at the same external times. The so produced data can be used to determine the relative time differences between the devices.

For the test bench setup, three test devices are used in a local network realized by a simple commercially available router for end users. Also part of the setup are two Raspberry Pis. One is used as a time server in the local network (the server our test devices synchronize their time with) and the other is placed between the router and the time server Raspberry Pi to simulate various effects of an unreliable network such as delays or package loss. This "disturber" Raspberry Pi is equipped with two network interfaces that are operated in a software bridge configuration. The network troubles are induced by the Linux kernel's traffic control capabilities that are configured using the tool `tc`[1].
The established network for time synchronization is additionally leveraged to collect the timestamp recordings. The test devices send the timestamps they saved upon the occurrence of a pulse from the signal source through the network to a server that collects them and is thus capable of carrying out statistical analyses.

5.1.1 Preliminary Experiments

In the first attempts to design the test bench, as illustrated in Figure 5.1, the test devices were connected via WiFi. Over this connection, they synchronized with the time server and reported the times recorded on pulse occurrence.
Experiments with this setup showed rather big variations in achieved synchronization results on different days. The experiments were performed in a residential area with more than 10 access points nearby, and the big variations in synchronization results are attributed to the typical problems of wireless connections like changes in transmission time and variable packet loss due to different levels of activity of surrounding access points, people moving in the vicinity of or test setup, etc.
So although this work is about time synchronization over WiFi and should account for the problems that typically occur when using WiFi, actually using WiFi in the test bench distorts the measurements in a way that makes them incomparable to other measurements performed in the same setup. This makes the use of WiFi unsuitable for the test bench.

[1] `https://man7.org/linux/man-pages/man8/tc.8.html`.

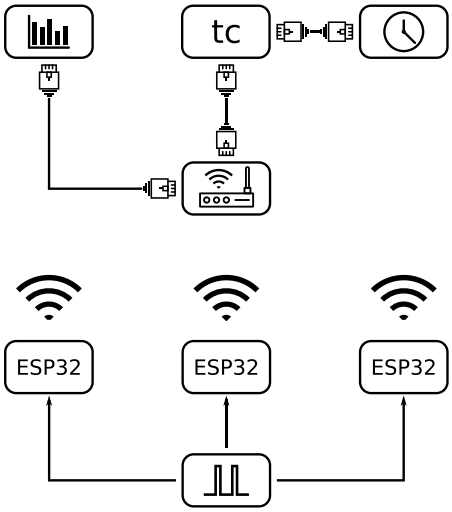


Figure 5.1: Preliminary test bench setup. The ESP32s are connected to a signal source (bottom center) and use WiFi to synchronize with the time server (top right) as well as to send their time measurements to the corresponding application (top left). The `tc` component on the path between the microcontrollers and the time server is used to simulate different network conditions.

Detecting Signal Source Pulses accurately and reliably is a core requirement for the microcontrollers in the test bench. The signal source in the test bench is configured to produce a Pulse Width Modulation (PWM) signal with a period of two seconds and a pulse width of one millisecond. This signal is used to coordinate time measurements between the microcontrollers. They are wired to the signal source and should report their local time on each signal pulse. This way time offsets between the microcontrollers can be determined thus information on the synchronization quality can be obtained.
The first approach to detect these pulses with the microcontroller was to connect the signal source to one microcontroller pin and register an interrupt on the rising edge for that pin. During initial tests with this configuration, pulses were properly detected. However, problems arose during the first longer-running tests with three microcontrollers. Since the signal source produces a pulse of one millisecond every two seconds and every microcontroller should report the local time on every pulse, the expected number of time measurements per microcontroller is one every two seconds. Sometimes, however, one of the microcontrollers reported an additional measurement in this two-second window. This issue was not limited to a single microcontroller and sometimes appeared after a few minutes, sometimes only after a few days.
Due to how the collection of the measurements in the test bench is realized, the detection and removal of these additional measurements is not trivial. Hence attempts were made to prevent the "ghost interrupts" that caused the additional measurements from happening. The first approach for improvement was to not only react to rising edges only but to react on both, rising and falling edges and, given the known pulse width, determine whether

```
void IRAM_ATTR gpio_isr_handler(void* args) {
  static int64_t last = 0;

  if(last == 0) {
    last = esp_timer_get_time();
  } else {
    int64_t diff = esp_timer_get_time() - last;
    last = 0;
    if(diff >= PULSE_WIDTH_MS*1000-500 &&
    diff <= PULSE_WIDTH_MS*1000+500) {
      BaseType_t xHigherPriorityTaskWoken = pdFALSE;
      xSemaphoreGiveFromISR(binarySemaphore,
      &xHigherPriorityTaskWoken);
      portYIELD_FROM_ISR_ARG(xHigherPriorityTaskWoken);
    }
  }
}
```

Listing 5.1: Interrupt Service Routine for both, rising and falling edges of the signal source.

or not this pulse was a "proper" one. As shown in Listing 5.2, the implementation of this approach expected two edges per pulse, a rising and a falling one. When using this approach, sometimes one of the microcontrollers stopped sending measurements. Our hypothesis is that the "ghost interrupts" were super short pulses so the microcontroller was not able to properly detect a rising and a falling edge for those. If once only a rising edge is detected, this implementation cannot work properly.

The peculiarity of this problem is that all interrupt pins of the microcontrollers are connected to the same signal source and yet one of them recognizes an edge while the others do not. Upon inspecting the power supply of the microcontrollers it was discovered that the power supply used up until now was extremely noisy with a peak-to-peak voltage of up to 15V (Figure 5.2).

So to work the problem the power supply was replaced with a way less noisy one (about 0.4V Vpp) and the pulse detection was changed. To improve the detection, a second microcontroller pin was connected to the signal source. One pin was used to register an interrupt for the positive edge, on the other pin an interrupt for the negative edge was registered. Upon receiving the positive edge interrupt, the microcontroller makes note of the time. When the negative edge is received, the microcontroller checks if the positive edge was registered roughly one millisecond before and only then triggers a measurement. This should ensure that there is no possibility that eventual short pulses can prevent any further measurements.

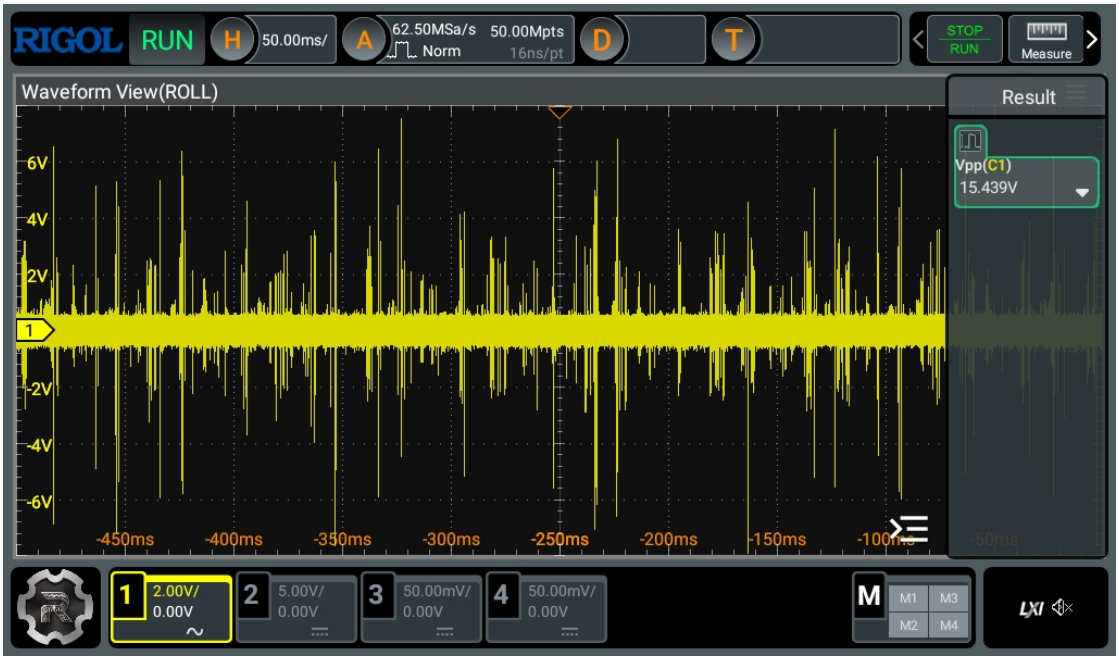


Figure 5.2: The noise of the first power supply used in the setup, possibly the source of the experienced ghost interrupts on the microcontroller.

With the combination of a way less noisy power supply and the two-pin interrupt pulse detection, no more interrupt-related problems were encountered during the use of the test bench.

```
static volatile int64_t last_pos_edge = 0;

void IRAM_ATTR gpio_isr_handler_pos_edge(void* args) {
  last_pos_edge = esp_timer_get_time();
}

void IRAM_ATTR gpio_isr_handler_neg_edge(void* args) {
  int64_t diff = esp_timer_get_time() - last_pos_edge;
  if(diff >= PULSE_WIDTH_MS*1000-200 &&
  diff <= PULSE_WIDTH_MS*1000+200) {
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    xSemaphoreGiveFromISR(binarySemaphore,
    &xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR_ARG(xHigherPriorityTaskWoken);
  }
}
```

Listing 5.2: Dedicated Interrupt Service Routing for Rising and Falling Edges.

5.1.2 Test Bench Architecture

Preliminary experiments (subsection 5.1.1) show that using WiFi in our test bench makes it hard to impossible to recreate measurement conditions. However, reproducible conditions are needed to test different synchronization approaches and obtain results that are comparable with each other.
Our solution approach for this problem is to remove the WiFi component and connect our test devices via Ethernet. The wired connection is expected to provide a way more consistent environment to test different approaches and obtain comparable results. Factors like changes in transmission time and variable packet loss, which are typical issues in wireless networks, can be simulated in the wired test bench using `tc` on the path between the microcontrollers and the time server. Modules based on the WIZnet W5500 chip were used to connect the microcontrollers via Ethernet.

As illustrated in Figure 5.3, within the test bench setup there are five wired connections. Because the available router featured only four Ethernet ports, a five-port switch was added to the setup. As illustrated in Figure 5.4, the three test devices and the `tc` component are connected to the switch to ensure that the network path for the synchronization packets is identical for all three test devices. To prevent potential biases, the test devices are connected using network cables with the same category, length, and manufacturer.

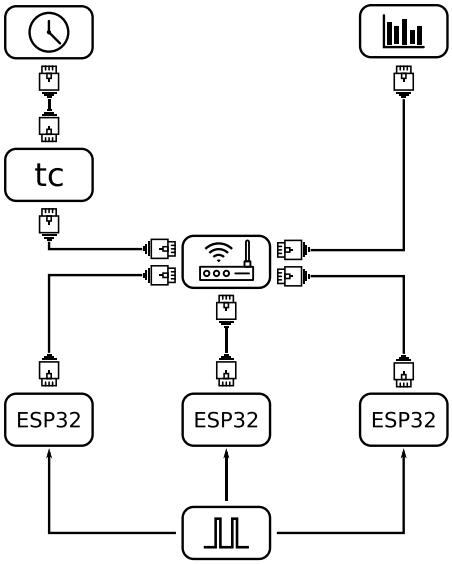


Figure 5.3: Wired test bench architecture. The ESP32s are connected to a signal source (bottom center) and to a router (center). The time server (top left) is connected to the router through the `tc` component, which is used to simulate different network conditions. The application that receives the measurements from the microcontrollers (top left) is also connected to the router.

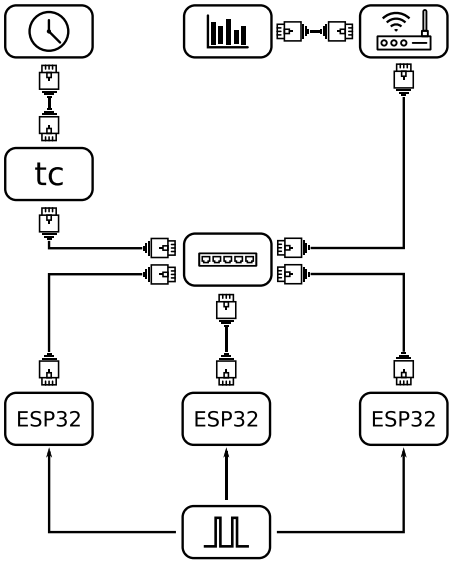


Figure 5.4: Final test bench architecture. A signal source (bottom center) is connected to the ESP32s. Those are connected to a switch (center). The time server (top left) is connected to the switch through the `tc` component, that is used to simulate different network conditions. The switch is connected to the router (top right). The application that receives the time measurements from the microcontrollers (top center) is connected to the router.

5.1.3 Microcontroller Firmware

In our test bench, the microcontrollers have multiple tasks. The first task is to synchronize the local time with the time server in the test bench at regular intervals. The second task is to detect pulses from the signal source and take note of the local time at which pulses are detected. These time values are called time measurements. Finally, the third task is to send the time measurements to the application in our test bench that collects them.

The software development kit for the ESP32, provided by its manufacturer Espressif, comes with FreeRTOS, an open-source real-time operating system. The implementation of the microcontroller firmware leverages the possibilities offered by the FreeRTOS and is organized in FreeRTOS `Tasks`[2].

Figure 5.5 shows the structure of the microcontroller firmware. The interrupt service routine (`ISR`), as discussed in section 5.1.1, upon successful detection of a pulse on the signal source notifies the `Time Measurement Task` via a FreeRTOS Semaphore, a popular construct to coordinate tasks in multi-threaded programming. The `Time Measurement Task` retrieves the current local time and pushes that value via a FreeRTOS Queue to the `Send To Backend Task`, which then sends this value via an open Transmission Control Protocol (TCP) socket connection to the measurement receiving application in the test bench. The `Time Synchronization Task` does not directly communicate with the other `Tasks` but modifies the local time, which can be seen by the `Time Measurement Task` when retrieving the current local time.
Upon starting, the `Send to Backend Task` opens the TCP connection to the measurement receiving application by connecting to the application and sending the Media Access Control (MAC) address of the microcontroller. For easy readability in debugger, network traffic analyzer, etc., the MAC address and following time measurements are transmitted ASCII encoded with a tailing newline character (`\n`, 0x0a).

[2] `https://www.freertos.org/Documentation/02-Kernel/02-Kernel-features/01-Tasks-and-co-routines/00-Tasks-and-co-routines`, accessed on 2024-09-29

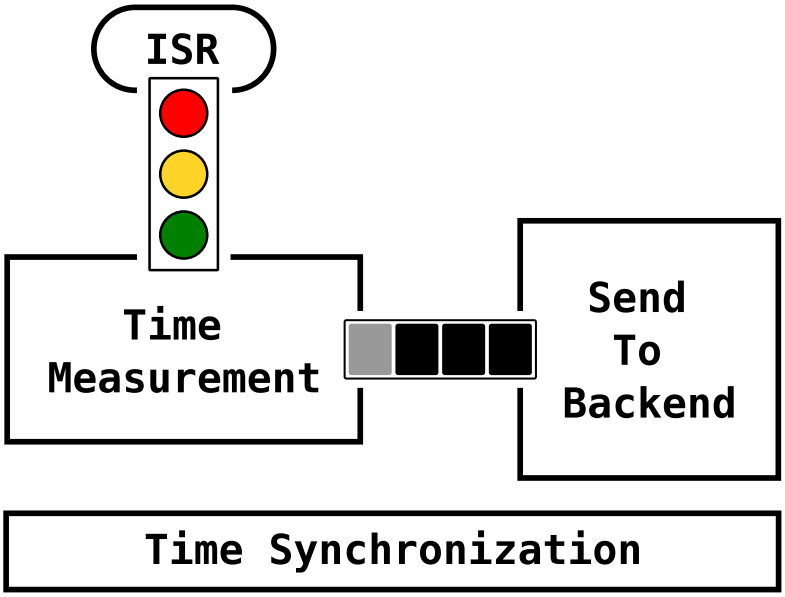


Figure 5.5: Structure of the software running on the microcontrollers in our test bench. The interrupt service routine (`ISR`) detects a pulse and triggers the `Time Measurement Task` via a Semaphore. The `Time Measurement Task` retrieves the local time and passes it via a Queue to the `Send To Backend Task`, which sends the measurement to the application collecting the measurements. The `Time Synchronization Task` runs independently.

5.1.4 Measurement Collection

The measurements taken by the microcontrollers in our test bench need to be collected for statistical analysis. For this task, the already existing network for time synchronization is leveraged. Every microcontroller sends its measurements to a central point in our setup, the Measurement Receiving Application (MRA).
This application opens a Transmission Control Protocol (TCP) server socket to which the microcontrollers can connect. As described in subsection 5.1.3, every microcontroller opens a TCP connection to the MRA and sends its MAC address. The MRA contains a list of MAC addresses it expects to connect and waits for all of them before starting to process the measurements sent by the microcontrollers. Processing the measurements means waiting until there is one measurement available for each microcontroller and printing those, along with the current timestamp and the maximal deviation between the three timestamps from the microcontroller. This is printed to both the standard output of the application and a Comma-separated values (CSV) file, a text file format used to store structured data. (Listing 5.3, lines 4, 6, 8, 10)
This approach is based on the assumption that all microcontrollers roughly start at the same time, connect to the MRA, send their MAC address roughly at the same time, and only afterward start to transmit time measurements in the form of timestamps. For

```
1  connected device 1 (xx:xx:xx:xx:xx:5c)
2  connected device 2 (xx:xx:xx:xx:xx:6c)
3  connected device 3 (xx:xx:xx:xx:xx:c8)
4  [02/09/2024 13:16:11] diff: 220 (1725275771130706, 1725275771130742, 1725275771130522)
5  false, false, false
6  [02/09/2024 13:16:13] diff: 241 (1725275773130679, 1725275773130744, 1725275773130503)
7  false, false, false
8  [02/09/2024 13:16:15] diff: 258 (1725275775130660, 1725275775130777, 1725275775130519)
9  false, false, false
10 [02/09/2024 13:16:17] diff: 275 (1725275777130688, 1725275777130794, 1725275777130519)
11 false, false, false
```

Listing 5.3: Example Output to stdout of the Measurement Receiving Application.

the latter, it is important that all microcontrollers transmit exactly the same number of timestamps, with the single timestamps arriving at roughly the same time.
In the event of one microcontroller sending more measurements than others (see section 5.1.1) the results cannot be used to make statements about the performance of the synchronization of the microcontrollers in our test bench. To detect such behavior, the application prints debug information about if a microcontroller has sent a measurement that was not processed yet to the standard output of the application. (Listing 5.3, lines 5, 7, 9, 11)

5.1.5 Local Time Server

The test bench also contains a timerserver with which the microcontrollers synchronize their time. Since the baseline (section 5.2) synchronization approach is SNTP, the local time server needs to be capable of communicating using the on-wire protocol [RFC5905]. To fulfill this requirement, chrony, an open-source implementation of the Network Time Protocol (NTP) is used. Chrony synchronizes with NTP servers outside our test bench and acts as an NTP server for our microcontrollers.

5.1.6 Traffic Control

To ensure the reproducibility of measurements in our test bench, a wired network is used and the characteristics of wireless connections are simulated. From the application layer software perspective, these are either packet delay or packet loss. A Raspberry Pi with two network interfaces and `tc`, a Linux utility program to configure the traffic control features of the Linux kernel is used to drop or delay packets in the network (subsection 5.1.2).

Asymmetric Packet Delay

When first experimenting with the baseline implementation in the test bench, asymmetric delays were introduced by configuring traffic control to add a 5ms +-4ms normally distributed delay on the path from microcontrollers to the time server and a 1ms fixed delay on the path from the time server back to the microcontrollers. Besides asymmetric delay, this configuration should ensure that both paths between microcontrollers and

time server are affected by the overhead introduced by traffic control delaying the packets. After running the setup like this for about 48 hours, the resulting data showed a suspicious similarity to the results obtained when running the setup without additionally introduced delays (Figure 5.6). Our investigation of this matter revealed that the default configuration of lwIP, the component in the official ESP32 software development kit that contains the SNTP functionality that is used as the baseline approach in this work, has set `SNTP_COMP_ROUNDTRIP` to `0` in `sntp_opts.h`. This disables the round-trip delay compensation located in `sntp_process` in `sntp.c`, which means that the local time is simply set to the value of the transmit timestamp (`T3`) of the NTP message received from the NTP server. This is perfectly fine for SNTP, however, the specification encourages to implement the full on-wire protocol for SNTP [RFC5905].

This does not mean that the baseline implementation isn't susceptible to asymmetric delay but that only the delay on the path from the time server to the microcontroller has a direct influence on the synchronization result. So to evaluate the baseline implementation using different asymmetric transmission delays and receive meaningful data, the direction of the introduced delay needs to be switched: 1 ms fixed delay on the path from microcontrollers to time server and a variable delay on the path from time server to microcontrollers.

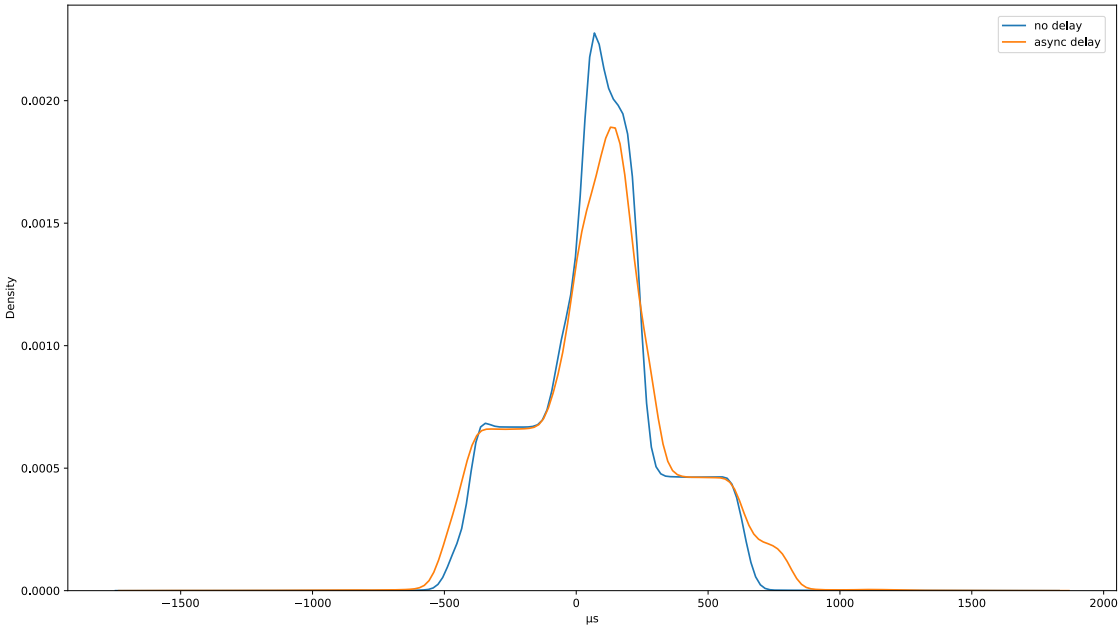


Figure 5.6: KDE plot showing the distribution of relative time differences without simulated delay and with simulated asynchronous delay on the path from the microcontrollers to the time server.

Varying the delay on one of the two paths is crucial because the metric for evaluating performance is the relative time difference between the microcontrollers. If all microcontrollers have their packets delayed for exactly the same amount, the relative difference

```
#eth0 time server to microcontroller
tc qdisc add dev eth0 root netem delay 5ms 1ms distribution normal
#eth1 uC to time server
tc qdisc add dev eth1 root netem delay 1ms
```

Listing 5.4: Traffic Control (`tc`) configuration for an asymmetic packet delay of 1ms jitter.

```
#eth0 time server to microcontroller
tc qdisc add dev eth0 root netem delay 10ms 7.5ms distribution normal
#eth1 uC to time server
tc qdisc add dev eth1 root netem delay 1ms
```

Listing 5.5: Traffic Control (`tc`) configuration for an asymmetic packet delay of 7.5ms jitter.

between the microcontrollers will be zero, no matter the amount of asymmetry. Using `tc`, the delay is specified as normally distributed with time (mean) and jitter (standard deviation) [JLHW11]. For the measurements the mean and standard deviation that describe the normal distribution are varied. For the results, the value of the mean is not important as it is shared by all microcontrollers, and by using relative differences they cancel each other out. However, the variation of the standard deviation should have consequences on the relative time differences. Listing 5.4 and Listing 5.5 show examples for `tc` configurations for 1ms and 7.5ms jitter.

Packet Loss

Within the test bench, packets are lost both on the path from the microcontrollers to the timer server and on the path from the time server to the microcontrollers. This is to ensure that both paths are equally affected by the overhead caused by traffic control dropping packets. So when reporting upon a measurement at N% packet loss this means that N% of the packets from the microcontrollers to the time server and N% of the packets from the time server to the microcontrollers are lost. Listing 5.6 shows how `tc` was configured for the measurement with 5% packet loss.

```
#eth0 time server to microcontroller
tc qdisc add dev eth0 root netem loss random 5
#eth1 uC to time server
tc qdisc add dev eth1 root netem loss random 5
```

Listing 5.6: Traffic Control (`tc`) configuration for a measurement with 5% packet loss in both directions.

5.2 Baseline Approach Using SNTP

To establish a synchronization quality baseline, the default time synchronization mechanism that comes with the Espressif IoT Development Framework (ESP-IDF), the official development framework for Espressif chips such as the ESP32, is used. The ESP-IDF uses lwIP[3], a lightweight TCP/IP stack with a focus on resource constraint hardware. Besides others, lwIP offers an SNTP library that is also used by the ESP-IDF[4].
Listing 5.7 shows how SNTP is configured in the test bench setup to establish the baseline.

```
esp_sntp_config_t config = {
    .smooth_sync = false,
    .server_from_dhcp = false,
    .wait_for_sync = true,
    .start = true,
    .sync_cb = sntp_sync_callback,
    .renew_servers_after_new_IP = false,
    .ip_event_to_renew = static_cast<ip_event_t>(0),
    .index_of_first_server = 0,
    .num_of_servers = 1,
    .servers = {"192.168.33.33"}
};
esp_netif_sntp_init(&config);
sntp_set_sync_interval(60*1000); //sync evey 60 seconds
esp_netif_sntp_start();
```

Listing 5.7: The SNTP configuration used to establish the synchronization performance baseline.

[3] https://savannah.nongnu.org/projects/lwip/, accessed on 2024-09-29
[4] https://docs.espressif.com/projects/esp-idf/en/release-v5.1/esp32/api-reference/system/system_time.html, accessed on 2024-09-29

5.3 Proposed Approach RAMSES

To improve upon the Baseline, the Baseline data was analyzed to get a picture of the issues there are. The findings regarding the Baseline results are presented in section 6.1 and show that in order to improve, the proposed approach should address the increasing standard deviation of the result data with increasing asymmetric transmission delays as well as the increasing numbers of outliers with increasing packet loss.
Typically, when a higher accuracy than (S)NTP is needed, some form of low-level, usually hardware timestamping is used [KK23]. However, for the sake of portability, an application layer software approach is pursued in this work. The proposed approach is named RAMSES, which stands for **R**epeated **A**veraging of **M**ultiple **S**NTP **E**xecutions for **S**ynchronization.

Transmission delays result from signals having to penetrate different materials due to environmental changes, signal interferences, reflections, and so on [HHN+07]. For a transmission delay to be asymmetric, corresponding changes must occur in the small time window between the transmitting and receiving parts of the transmission. If this is not the case, both parts would be subject to the same transmission delay and there would be no asymmetry. So in the context of wearable devices with wireless communication, it is assumed that the causes for asymmetric transmission delays are of rather short duration and a random nature. The latter is assumed for the sake of creating a mental model of the issue because although there are certainly reasons for asymmetric transmission delays, application layer software has no way of determining them and can only consider asymmetric delays to be random.
To lessen the effects of asymmetric transmission delays on the time synchronization result, multiple on-wire protocol results that each contain some amount of error are collected and it is attempted to find the hidden, error-free time difference value through statistical means. With this approach, the more on-wire results are used for synchronization, the better the result will be. However, the more on-wire results are used, the more network communication and computing power are required to perform the on-wire protocol, which is at the expense of energy consumption and the runtime for other tasks that the microcontroller has to perform. Therefore, RAMSES uses five on-wire results at one-second intervals. Waiting for some time between the on-wire protocol executions follows from the assumption that causes for transmission delays are of rather short duration. Compared to performing all on-wire protocol executions as fast as possible after one another, waiting between the on-wire protocol executions should increase the chance that if one execution is affected by an asymmetric transmission delay, the others will remain unaffected. While some waiting is beneficial, when waiting for too long the clock drift of the microcontrollers might have a non-negligible impact. This could lead to a situation where the collected on-wire protocol results are not distributed around the "true" offset anymore but rather follow a trend defined by the clock drift of the device.
To answer the question of how to best evaluate the collected on-wire protocol results, 100000 on-wire protocol results were simulated. All of the on-wire protocol executions share the same "true" offset, however, each one has a difference in transmission times

somewhere between no difference and 6ms. Using a moving window of size five to go through the data, different statistical methods were applied to the data in the window. This results in 99996[5] values per applied statistical method. The closer the values of a method are to the "true" offset, the better the method fits our needs. The performance of the different applied methods can be seen in Figure 5.7 and Table 5.1. There are four methods that performed best. Notably, the mean and standard deviation of their results are all equal. The four methods are mean, linear regression, ARD regressor, and Tweedie regressor. The implementation of the mean is straightforward and most resource-efficient. This makes it a practical choice for a resource-constrained device and thus the mean is used in RAMSES.

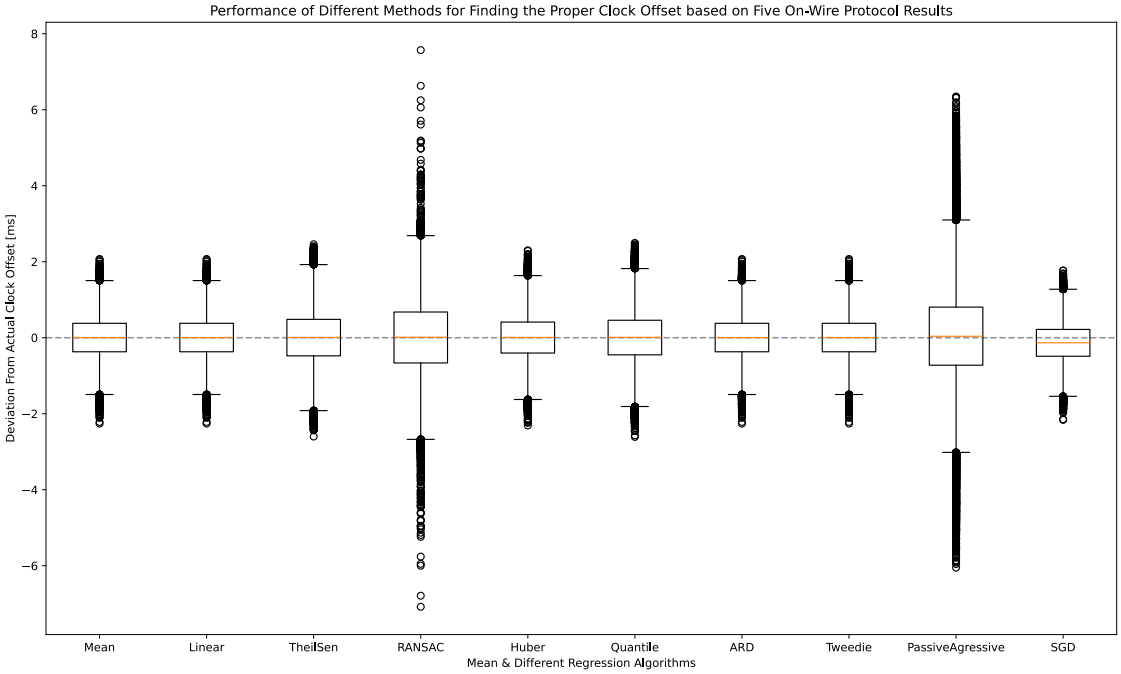


Figure 5.7: Performance of different statistical methods when determining "true" time offset (centered at zero) using multiple on-wire protocol results.

Collecting and averaging five on-wire results at one-second intervals is an approach to improve synchronization performance when synchronization is affected by asymmetric transmission delays. While this approach will also dampen the effects of outliers, especially the extreme outliers of one and two seconds found in the Baseline results (section 6.5) will still have a significant influence on the synchronization result. For example when four of five on-wire results are around a few milliseconds and one is around one second, a

[5]When using a moving window of size m over a dataset of size n, the first window that is applied covers the first m entries 1 to m, the second window covers entries 2 to $m+1$, etc. until the last window covers the entries $(n-m+1)$ to n. The window starts at the beginning of the dataset and is continuously moved one position further until the end is reached. Consequently, the starting position of the last window equals the count of how often a window was applied. This is $n-m+1$.

Name	μ	σ	Method
Mean	0.003	0.548	Statistical Mean
Linear	0.003	0.548	Linear Regression
TheilSen	0.004	0.708	Theil Senn Regressor
RANSAC	0.005	1.000	RANSAC Regressor
Huber	0.004	0.599	Huber Regressor
Quantile	0.005	0.692	Quantile Regressor
ARD	0.003	0.548	ARD Regressor
Tweedie	0.003	0.548	Tweedie Regressor
PassiveAgressive	0.057	1.305	Passive Agressive Regressor
SGD	-0.135	0.510	SGD Regressor

Table 5.1: Mean and Standard Deviation of the data displayed in Figure 5.7.

simple mean of these five values would produce a value of around 200ms.
To face the outliers that come with increasing packet loss, they must thus be detected and discarded before the mean value is calculated. The strategy thus changes to taking the mean of five on-wire results that are not outliers. If an outlier is detected, it gets discarded and the on-wire protocol is re-executed immediately until a non-outlier result is obtained.
Due to different clock drift rates caused by changing environments, "proper" synchronization result values can have different baselines. Therefore, outliers cannot be detected using numerical thresholds. In contrast to baseline shifts, however, outliers are of a sudden and non-persistent nature. So a low-pass filter is used to detect them. Every on-wire result updates the filter so that long-term trends are respected, while short spikes are rejected. Equation 5.1 shows the implemented filter. In addition to the low-pass filter, the impact of a single on-wire result is limited by capping the value as seen in Equation 5.2. To keep things simple, only absolute values are used. This way, the filter value can be interpreted as the "value by which the local time is usually corrected". Based on this value, outliers can be detected. How much an on-wire result can deviate from this value before it is considered an outlier is a factor that can be fine-tuned. Generally speaking, allowing for more deviation causes more scatter of synchronization results while allowing for fewer deviations increases the amount of synchronization processes because more results get discarded as outliers. In the RAMSES implementation, up to 20% more than the filter value is allowed before a result is considered an outlier (Equation 5.3).

$$f_n = |f_{n-1} \times 0.95 + u_n \times 0.05| \tag{5.1}$$

$$u_n = \begin{cases} min(|m|, f_{n-1} \times 10), & \text{if } f_{n-1} > 0 \\ |m|, & \text{otherwise} \end{cases} \tag{5.2}$$

$$outlier\_threshold = |f_n| \times 1.2 \tag{5.3}$$

f_n : filter value after n filter updates
u_n : the n-th update value for the filter
m : the measurement with which the filter should be updated. In our case, this is the result of the on-wire protocol

In Figure 5.8 the behavior of the discussed filter is illustrated. For this illustration measurement values with outliers were simulated. The filter starts with a value of 0. The factors 0.95 for the previous filter value and 0.05 for the update value ensure that the filter slowly adapts to changes. Consequently, the effects of sudden, temporary changes are strongly dampened. Because of this slow adaption, however, especially at the beginning, the filter needs some measurements before its value reaches the level of the measurement values. Assuming the start value for the filter to be 0 and a constant update value u_n, the filter value after n steps can be calculated:

$$f_n = f_0 \times (0.95^n) + u_n \times (1 - 0.95^n) \tag{5.4}$$

Since $f_0 = 0$ at the beginning, the equation can be simplified to

$$f_n = u_n \times (1 - 0.95^n) \tag{5.5}$$

With u_n assumed to be constant, it can be calculated how much percent of u_n the filter value equals after n updates:

n	u_n
1	5%
10	40%
20	64%
35	83%
50	92%
75	97%

Table 5.2: Percentage of u_n retained by the filter after n filter updates, assuming the filter value starts at 0 and u_n remains constant.

Table 5.2 shows an asymptotic approximation of f_n to u_n and that about 30 measurements are needed to reach about 80% of the measurement value. While this may seem quite a lot at first, this is necessary to properly react to extreme outliers. Figure 5.8 shows simulated measurements of around 2ms, that change to around 5ms towards the middle of the visualization and back to 2ms towards the end. Every now and then the simulated measurements contain an extreme outlier of about 1.5s. The slow adaption of the filter due to Equation 5.1 and the capping of the update value (Equation 5.2) lead to the filter value only increasing about 1-2ms when encountering extreme outlier measurements of an order of magnitude more than the usual measurements. To prevent update blocks and to ensure that the filter eventually adapts to long-term changes, all measurements, including outliers, are factored into the filter value. However, when it comes to calculating the mean of five on-wire results, outliers must be excluded, as they would distort the result.

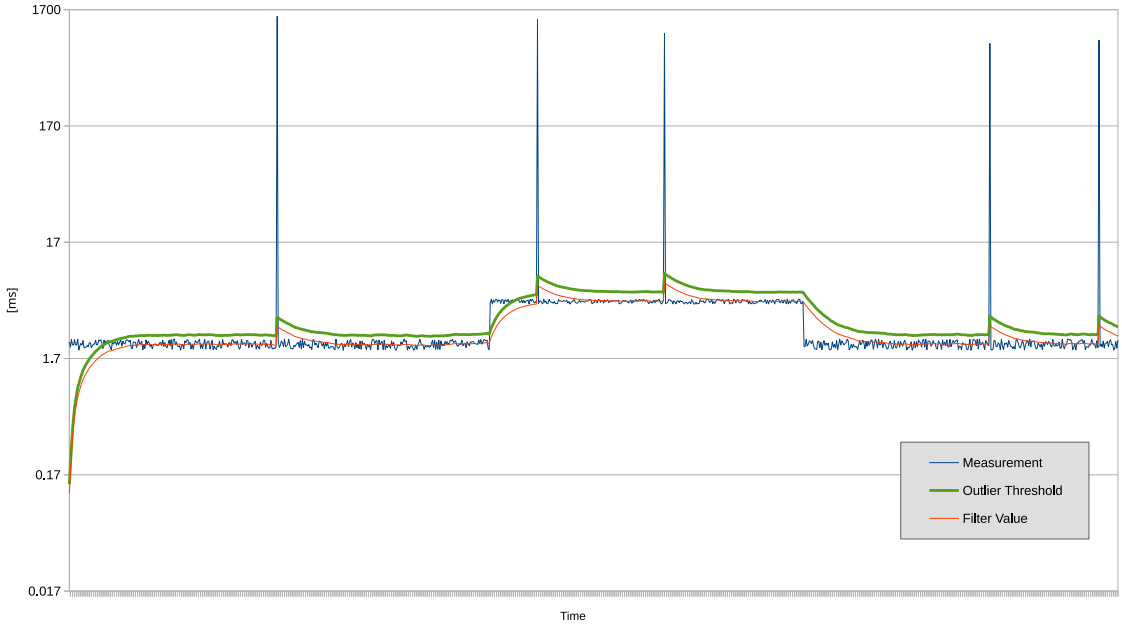


Figure 5.8: Behavior visualization of the filter used to detect outlier.

CHAPTER 6

Evaluation

In this chapter, the test bench results of two different time synchronization approaches, Baseline and RAMSES, are reported and evaluated. For both approaches, different amounts of asymmetric packet delay and different percentages of packet loss were simulated independently. Long-term measurements of about 96 hours were conducted to build statistically robust models of the synchronization performance. Short-term measurements of about 1 hour were carried out to validate the models, to understand their significance to shorter, more typical time periods for experiments with wearables, and to possibly identify systematic errors in the long-term measurements caused by periodic patterns such as the day/night cycle or similar factors. Finally, about 3-hour-long measurements were performed with two different combinations of packet delay and packet loss for both approaches. These combined measurements are intended to show the difference between the two approaches in more real-world scenarios.
Some of the simulated network parameters cause some test bench devices to synchronize later than others. This results in notable outliers in the data at the beginning of the measurements. Information about the synchronization performance can only be obtained from test bench measurements made after all devices have at least once properly synchronized with the time source. Therefore, outliers at the beginning of the measurement, which are due to the first synchronization of some devices being delayed, are unsuitable for assessing synchronization performance and are thus removed manually before the evaluation.

6.1 Baseline

The default time synchronization approach that comes with the Espressif IoT Development Framework (ESP-IDF) was used to set a baseline in terms of time synchronization performance. The ESP-IDF uses the SNTP functionality of the integrated lwIP TCP/IP

stack. Using the test bench, the behavior of this SNTP functionality was examined under varying packet loss and varying, asymmetric, packet transmission delay in the network.

6.1.1 Variable, Asymmetric Packet Delay

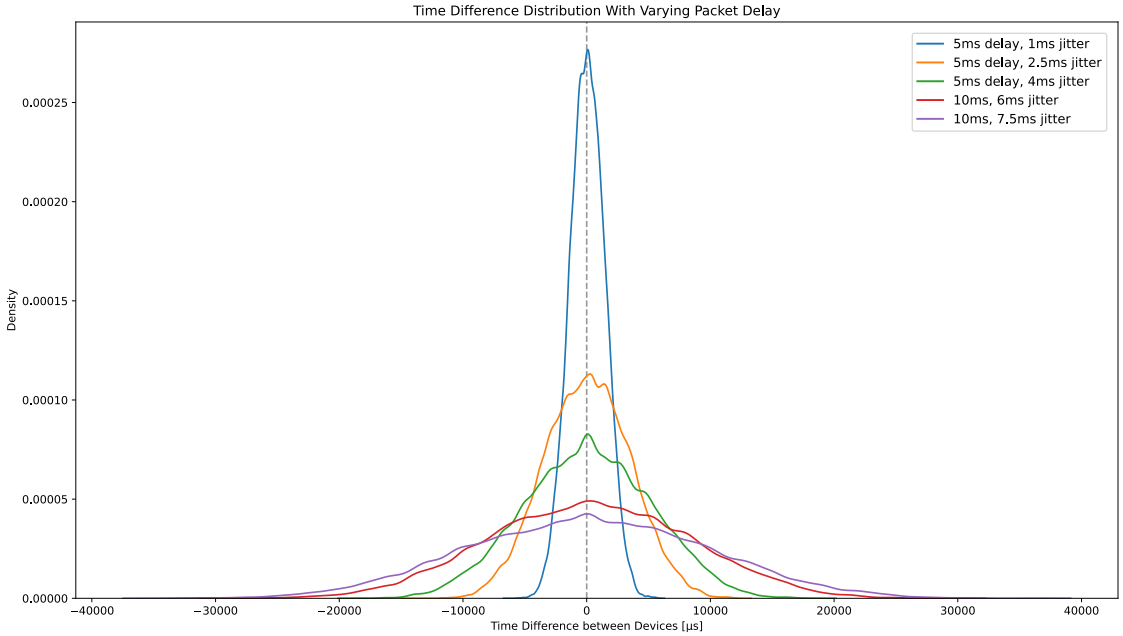


Figure 6.1: Distribution of relative time differences between devices at different packet delay jitter using the Baseline approach in the test setup.

Figure 6.1 shows how the synchronization results vary when changing the standard deviation (jitter) of the amount of asymmetric packet delay in the test bench. Note that in Figure 6.1 the mean value set in `tc` cannot be observed. This is because Figure 6.1 shows the distribution of relative time differences between the devices in the test bench. The delays applied to packets from each device all follow the same normal distribution and thus share the same mean. Due to the representation of relative differences, the distributions are centered around zero. Table 6.1 lists the mean and standard deviation of the data resulting from measurements with different jitter.

The values from Table 6.1 are plotted in Figure 6.2 and Figure 6.3 together with fitted linear functions to both the mean and the standard deviation. From looking at Figure 6.1, a higher delay jitter is expected to be connected to a higher standard deviation in the measurement data. This is confirmed by the fitted linear function $f(x) = 1.29x + 0.16$ for σ in milliseconds.
Figure 6.3 shows a fitted linear function $f(x) = 0.02x + 0.04$ for μ in milliseconds. However, the associated R^2 of 0.48 indicates that the fitted function does not fit the data particularly well. For each measurement, 174374 samples are collected. Because

Jitter [ms]	μ [ms]	σ [ms]
1	0.056	1.430
2.5	0.058	3.443
4	0.094	5.146
6	0.207	8.097
7.5	0.111	9.711

Table 6.1: Mean (μ) and standard deviation (σ) of the measurement data at different packet delay jitter using the Baseline approach. The jitter values were selected so that they go up to 7.5ms and are not multiples of each other.

one sample was taken every two seconds and time synchronization happened once per minute, this roughly equals 5812 time synchronization packets per device.

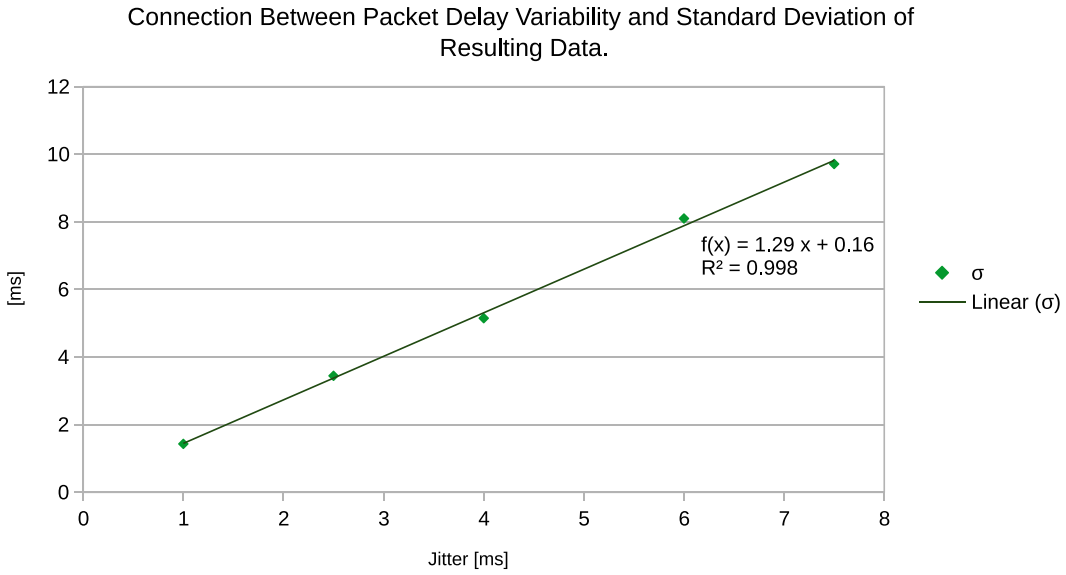


Figure 6.2: Measured standard deviation at different packet delay jitter using the Baseline approach with fitted linear function.

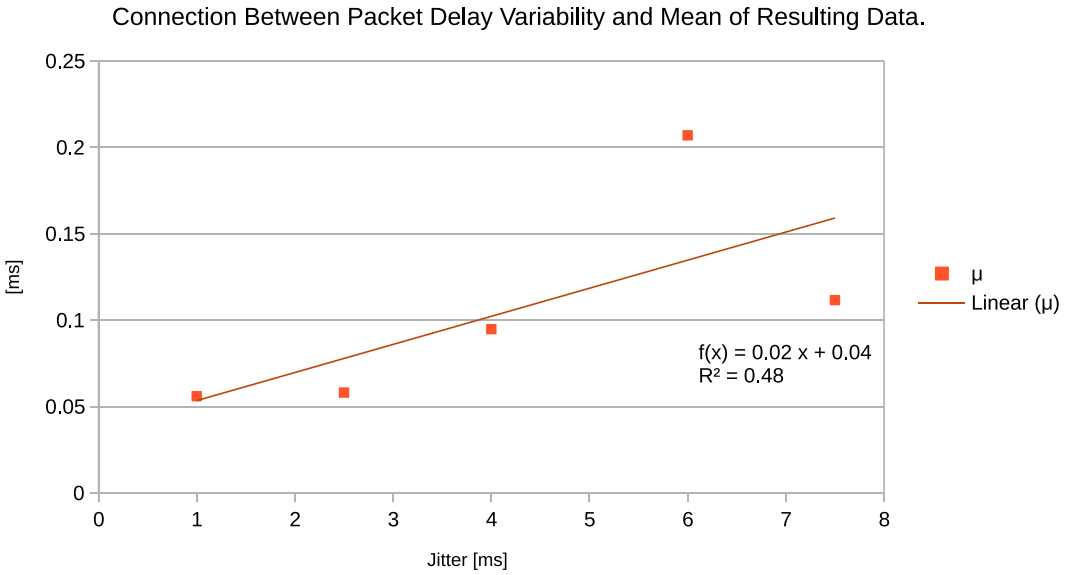


Figure 6.3: Measured mean at different packet delay jitter using the Baseline approach with fitted linear function.

Short-Term Performance

In Figure 6.4 the mean and standard deviation of the short-term measurements at different packet delay jitter values are plotted together with fitted linear functions. The resulting short-term function for the standard deviation (in milliseconds) is $f(x) = 1.273x + 0.272$ with an R^2 of 0.998 The fitted function for the mean is $f(x) = 0.018x - 0.192$ (in milliseconds) with $R^2 = 0.006$.

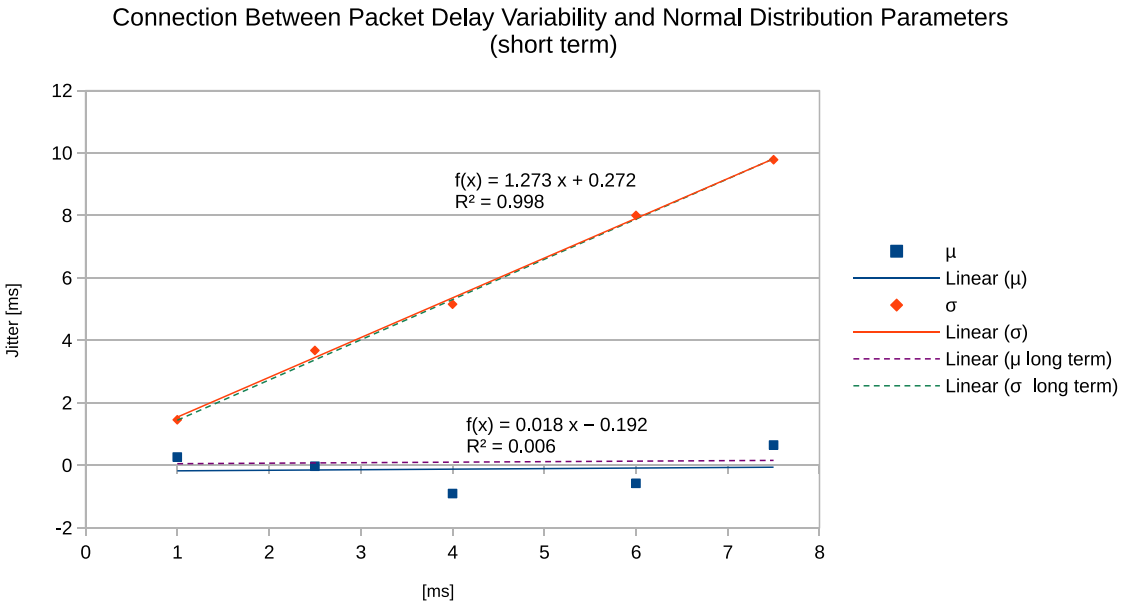


Figure 6.4: Connection between packet delay jitter and normal distribution parameters of the short-term measurement results obtained using the Baseline approach.

6.1.2 Variable Packet Loss

Using our test bench, measurements with different percentages of packet loss were conducted. The results are displayed as boxplots in Figure 6.5. No particular variation can be observed in the form of the different boxplots. However, starting with 7.5% packet loss, rather big outliers of about one second or more occur. Due to these outliers, the evaluation of the packet loss cannot be done in the same way as asymmetric packet delay. So for the packet loss evaluation, outliers are discussed separately, then they are removed and the data without outliers is evaluated analogous to the asymmetric packet delay evaluation.
Table 6.2 contains the minimum and maximum time difference values between devices measured at different percentages of packet losses in the system. In line with the observations from the boxplots, a sudden increase in extreme values is found starting with 7.5% packet loss.

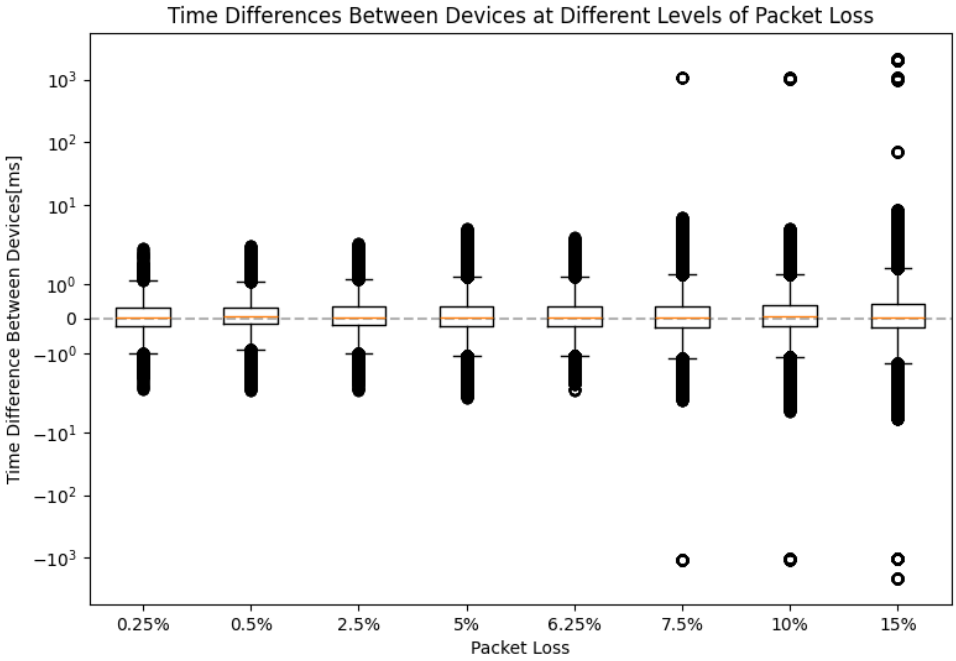


Figure 6.5: Boxplot of measured time differences between devices at different packet loss percentages using the Baseline approach.

Packet Loss	Min [ms]	Q1 [ms]	Q2 [ms]	Q3 [ms]	Max [ms]
0.25%	-2.03	-0.20	0.04	0.33	2.06
0.50%	-2.14	-0.15	0.09	0.33	2.29
2.50%	-2.13	-0.19	0.03	0.34	2.51
5.00%	-2.81	-0.21	0.04	0.35	4.19
6.25%	-2.15	-0.20	0.03	0.36	3.04
7.50%	-1070.27	-0.23	0.02	0.37	1051.89
10.00%	-1077.54	-0.21	0.06	0.38	1077.62
15.00%	-2121.51	-0.25	0.04	0.43	2121.48

Table 6.2: Minimum, quartiles 1-3 and maximum of the measured time difference between devices at different packet loss percentages using the Baseline approach.

While in Figure 6.5 and Table 6.2 big outliers of about one second or more can be seen, there is no information about smaller outliers. For this matter, the outliers are divided into two groups. All data points that are 50ms or more are referred to as *extreme outliers*. After removing these extreme outliers, the 1.5 times interquartile range rule is applied to determine the remaining, smaller outlier. Those are labeled *IQR*1.5*. The Number of outliers in each group at different percentages of packet loss are listed in Table 6.3.

These numbers confirm the previously observed extreme outliers that appear with packet loss of 7.5% or more. However, these numbers also show that the quantity of outliers in the datasets rises with increasing packet loss percentage. To get a better understanding of the relation between packet loss percentage and outlier quantity the values are plotted in Figure 6.6 with fitted a polynomial function of degree two onto the data. The R^2 value of 0.998 indicates that the function $f(x) = 123.98x^2 - 130.07x + 883.16$ fits the data very well, and thus it is assumed that the relation between packet loss percentage and outlier quantity is quadratic.

Packet Loss	Extreme Outlier (>50 ms)	Outlier (1.5*IQR)	Total	Percentage
0.25%	0	567	567	0.32%
0.50%	0	1172	1172	0.65%
2.50%	0	1317	1317	0.73%
5.00%	0	3754	3754	2.09%
6.25%	0	4284	4284	2.39%
7.50%	150	6800	6950	3.87%
10.00%	240	12323	12563	7.00%
15.00%	1170	26758	27928	15.56%

Table 6.3: Number of outliers at different packet loss percentages using the Baseline approach, determined by first removing extreme outliers (>50ms) and using the 1.5*IQR method to determine the number of outliers in the remaining data.

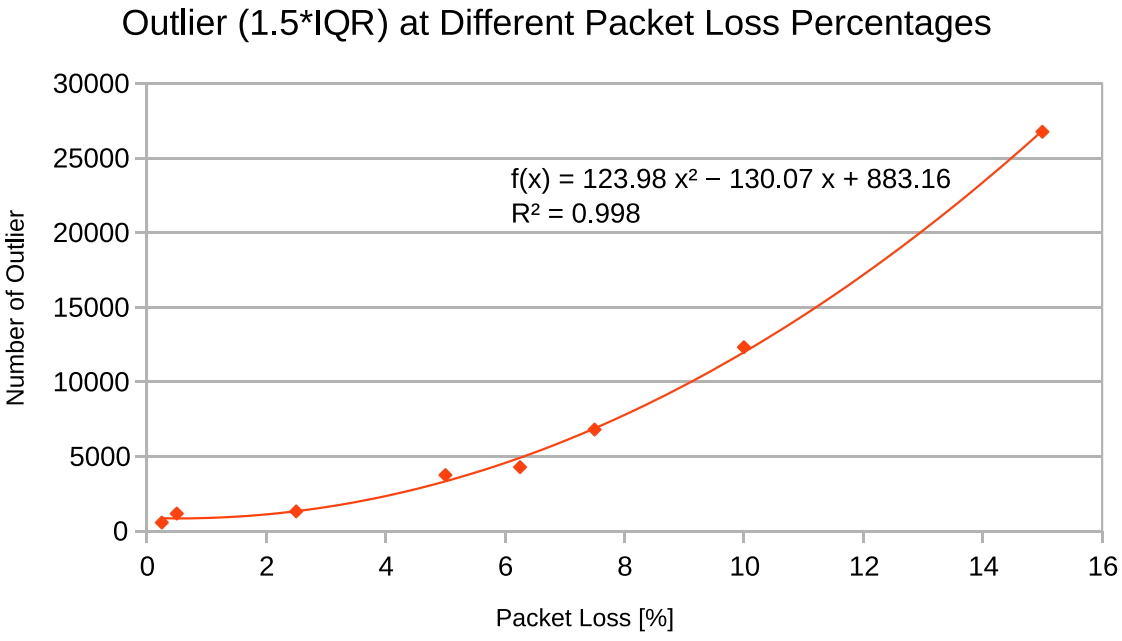


Figure 6.6: Number of outliers at different packet loss percentages using the Baseline approach with fitted polynomial function of degree two.

Besides minimum and maximum time difference between devices at different packet loss percentages, Table 6.2 also lists the quartiles 1-3 of the corresponding datasets. In Figure 6.7 the quartiles with fitted linear functions are plotted. The points for Q2 are poorly captured by the fitted linear function ($R^2 = 0.05$) and the function coefficients are rather small. Thus we argue that Q2 should be more of a constant value and the variation of Q2 in our measurements is predominantly because of the limited amount of collected data rather than because of the percentage of packet loss in the system. For Q1 and Q3 however, it can be observed that their values move away from each other with increasing packet loss in the system. When removing all outliers from the collected data and plotting the results at different packet loss percentages as KDE plot (Figure 6.8), a tendency towards increased curve widths with increasing packet loss can be seen.
Table 6.4 lists the mean and standard deviation values for the measurement results at different packet loss percentages without outliers. The values and fitted linear functions to mean and standard deviation are plotted in Figure 6.9. Also in this representation, it can be seen that the standard deviation increases with increasing packet loss percentage. This relation is not immediately apparent, as more packet loss should only cause more synchronization retries and should not directly affect the synchronization result per se because packets should either be lost or delivered without considerable delay. However, because with increasing packet loss more and more packets are lost and thus more and more retries have to be made, the average frequency of resynchronization will become lower. In the time between resynchronizations, the clocks of the microcontrollers drift. The lower the resynchronization frequency, the longer the microcontroller clocks are drifting and thus higher standard deviation values can be observed in the measurement results. To roughly estimate the drift of the test bench devices, a few manual samples were taken from the measurement results. It was found that the largest deviation between the devices is roughly around 12µs per second.
Note that the difference between the standard deviations at different packet loss percentages is in the tens of microseconds. So compared to the previously observed influence of asymmetric packet delay on the standard deviation of the measurement results, the influence of packet loss on the standard deviation of the results is an order of magnitude lower.

As for the mean, no pattern related to the packet loss percentage can be observed. The fitted linear function from Figure 6.9 does merely consider the packet loss value and the R^2 value of 0.012 indicates that the function captures the data points poorly. At this point, we repeat our argumentation for the 2nd quartile (median), which is that the mean is more of a constant value rather than dependent on the packet loss and that the different values at different packet loss percentages originate from the limited amount of data collected for our measurements.

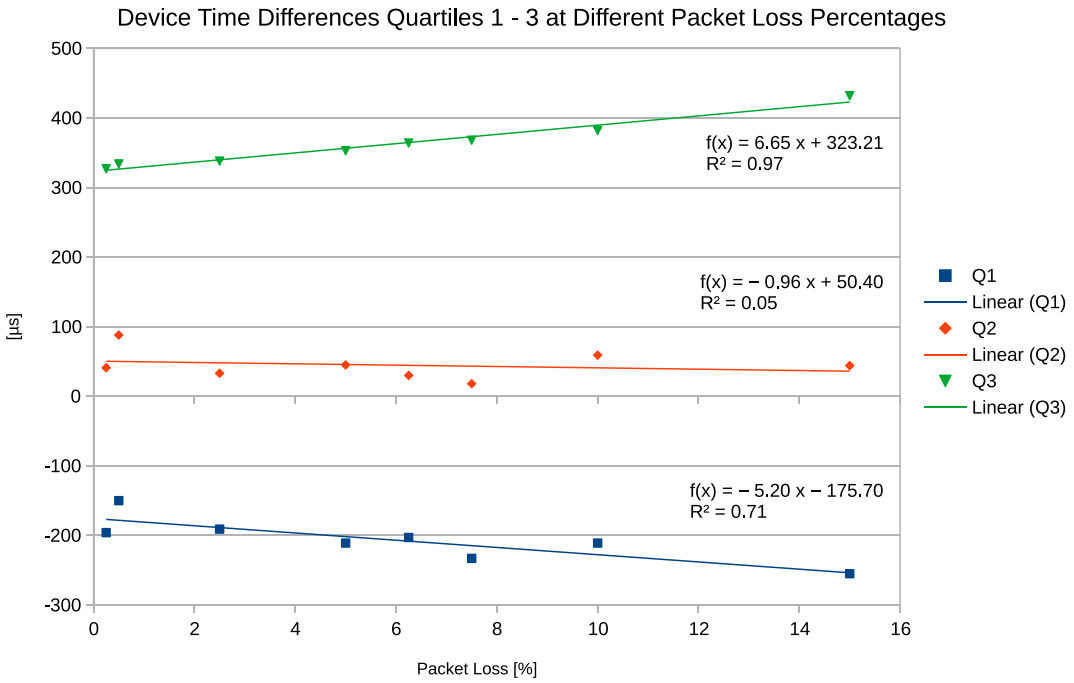


Figure 6.7: Measured quartiles at different packet loss percentages using the Baseline approach with fitted linear functions.

Packet Loss [%]	μ [μs]	σ [μs]
0.25	61.309	323.48
0.5	97.28	309.018
2.5	71.192	343.015
5	72.648	368.282
6.25	73.887	372.208
7.5	61.768	393.501
10	82.81	394.708
15	81.828	466.119

Table 6.4: Normal distribution parameters of measurement data obtained using the Baseline approach at different percentages of packet loss when outliers are removed. The packet loss percentages were chosen to cover both a large range of percentages and different step sizes between percentages. Small step sizes at lower percentages should capture the effects of small changes while larger step sizes at higher percentages should cover general trends in the synchronization performance.

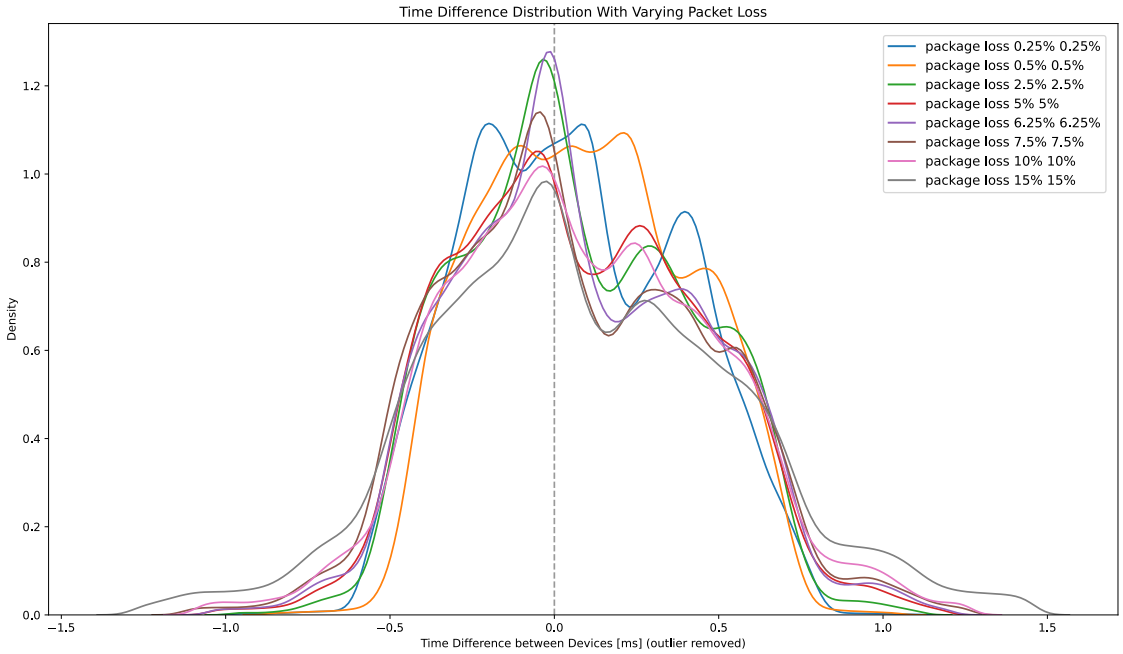


Figure 6.8: Distribution of relative time differences between devices at different packet loss percentages using the Baseline approach with outliers removed.

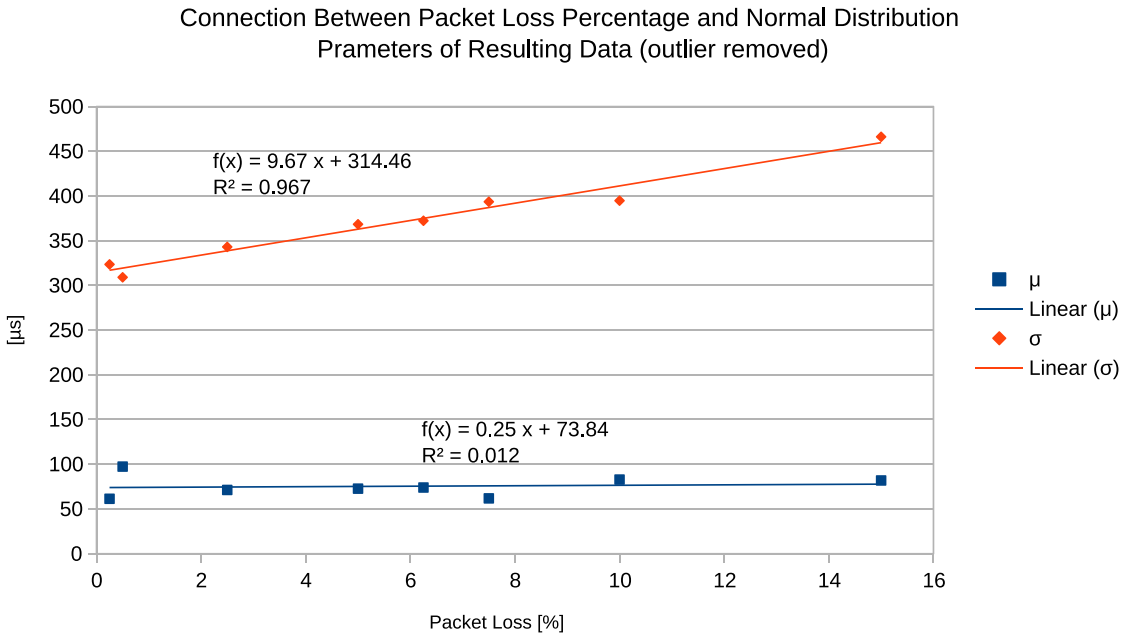


Figure 6.9: Measured mean and standard deviation with fitted linear functions after removal of outliers from the measurement data obtained at different packet loss percentages using the Baseline approach.

Short-Term Performance

In Figure 6.10 the mean and standard deviation of the short-term measurements at different packet loss percentage values are plotted with fitted linear functions. The resulting short-term function for the standard deviation (in microseconds) is $f(x) = 10.87x + 303$ with an R^2 of 0.87. The function for the mean value is $f(x) = -3.06x + 123$ (in microseconds) with $R^2 = 0.27$.

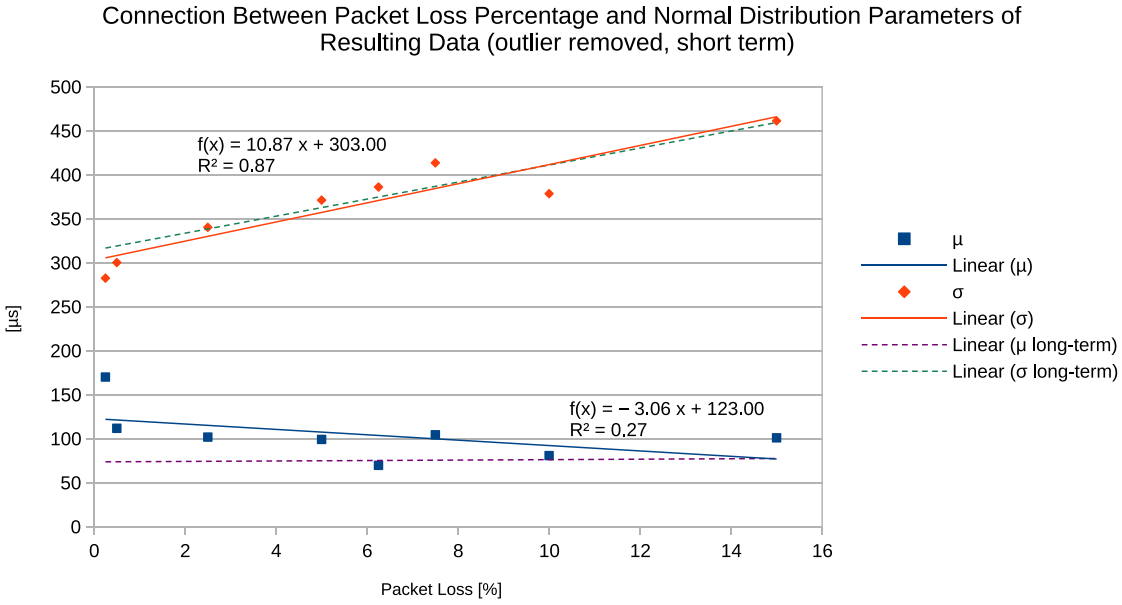


Figure 6.10: Connection between packet loss percentage and normal distribution parameters of the short-term measurement results obtained using the Baseline approach.

6.1.3 Baseline Evaluation Summary

The default time synchronization approach of the ESP-IDF is the SNTP functionality of the lwIP TCP/IP stack. This functionality was used to determine a baseline of time synchronization quality. To do so, different amounts of packet loss and asymmetric packet transmission delay were simulated on the test bench.

Our results show that the main issue with asymmetric transmission delays is a higher degree of data scattering. The standard deviation in milliseconds of our measurements is described by the function $f(x) = 1.29x + 0.16$ ($R^2 = 0.998$), with x being the jitter, that is the standard deviation of the random delay.

The main issue with packet loss on the other hand is the outliers that increase with the packet loss. The test bench results show the relation between the number of outliers and packet loss to be quadratic, described by the function $f(x) = 123.98x^2 - 130.08x + 884.16$ ($R^2 = 0.998$), with x being the packet loss in percent.

6.2 RAMSES

Using the test bench, the behavior of our proposed synchronization approach RAMSES was examined under varying packet loss and varying, asymmetric, packet transmission delay in the network.

6.2.1 Variable, Asymmetric Packet Delay

Figure 6.11 illustrates the distribution of the time differences between the devices at different jitter levels.

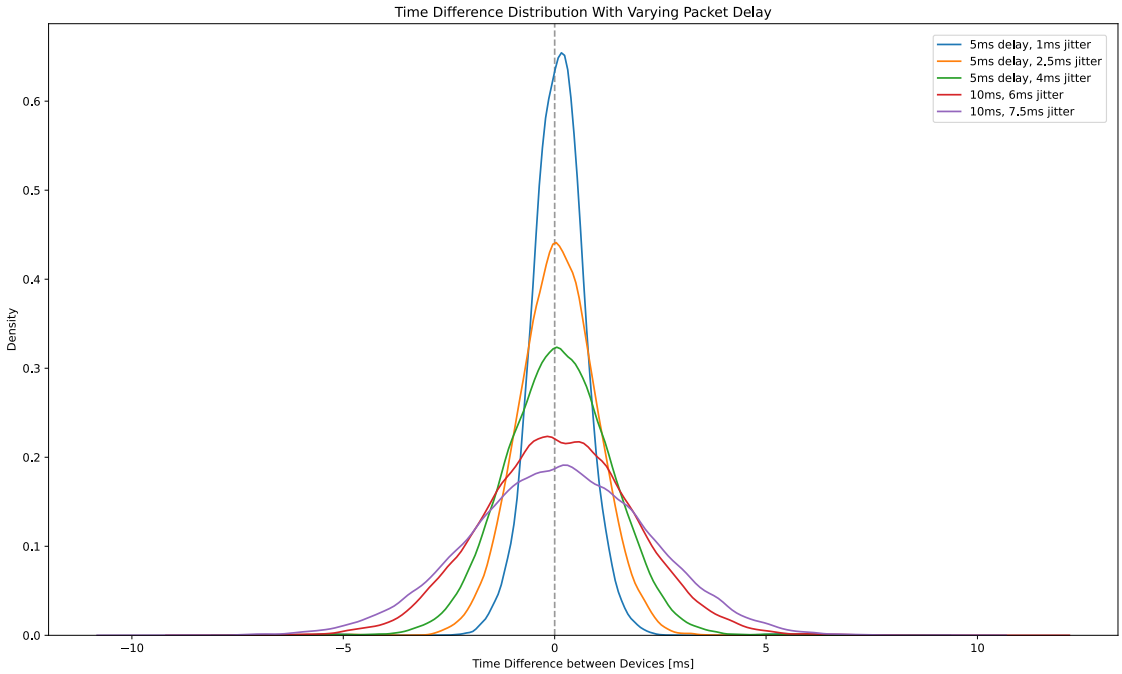


Figure 6.11: Distribution of relative time differences between devices at different packet delay jitter using RAMSES in the test setup.

Table 6.5 contains the mean and standard deviation of the data resulting from our measurements with different jitter when using the RAMSES approach.

Figure 6.12 and Figure 6.13 show the mean and standard deviation values from Table 6.5 respectively. Linear functions were fitted to those values. The fitted function for the standard deviation (in milliseconds) is $f(x) = 0.221x + 0.441$ with $R^2 = 0.995$. The function for the mean value (in milliseconds) is $f(x) = 0.001x + 0.087$ with $R^2 = 0.121$. A total of 180967 samples per measurement was collected.

While the fitted function for the standard deviation seems to be a good fit ($R^2 = 0.995$), the function for the mean has both a low factor for x (0.001) and a low R^2 value (0.121). Based on these values we argue that there is no relation between mean and jitter, but

Jitter [ms]	μ [ms]	σ [ms]
1	0.101	0.713
2.5	0.077	0.961
4	0.091	1.274
6	0.091	1.784
7.5	0.106	2.123

Table 6.5: Mean (μ) and standard deviation (σ) of the measurement data at different packet delay jitter using the RAMSES approach. The jitter values match the ones used for the Baseline approach. Those were selected so that they go up to 7.5ms and are not multiples of each other.

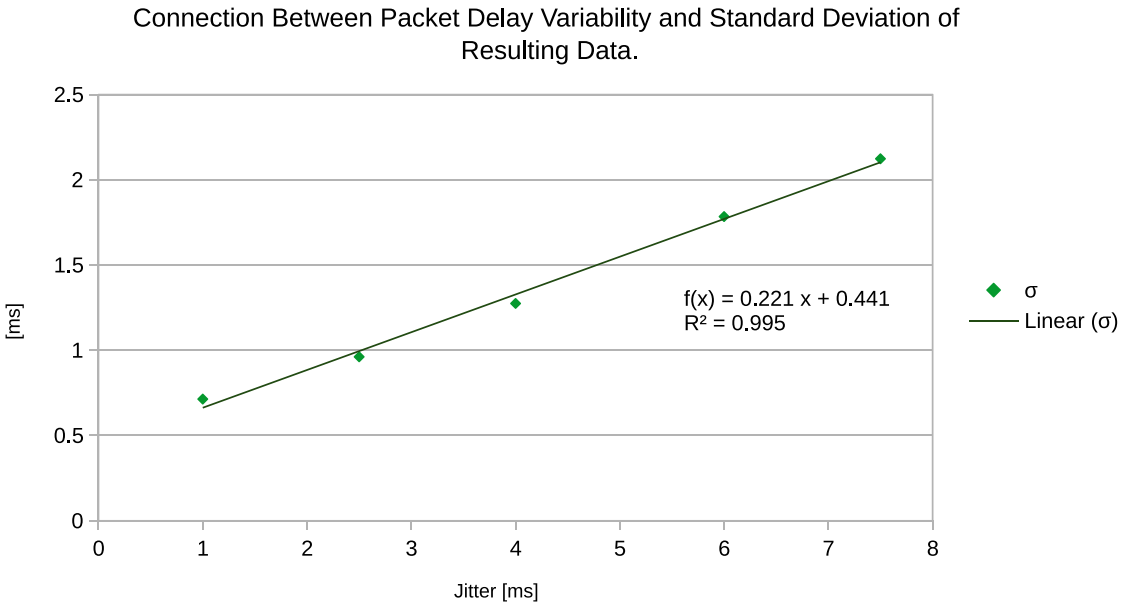


Figure 6.12: Measured standard deviation at different packet delay jitter using RAMSES with fitted linear function.

that the mean is rather constant. The different mean values observed at different jitter values are attributed to the limited amount of data collected.

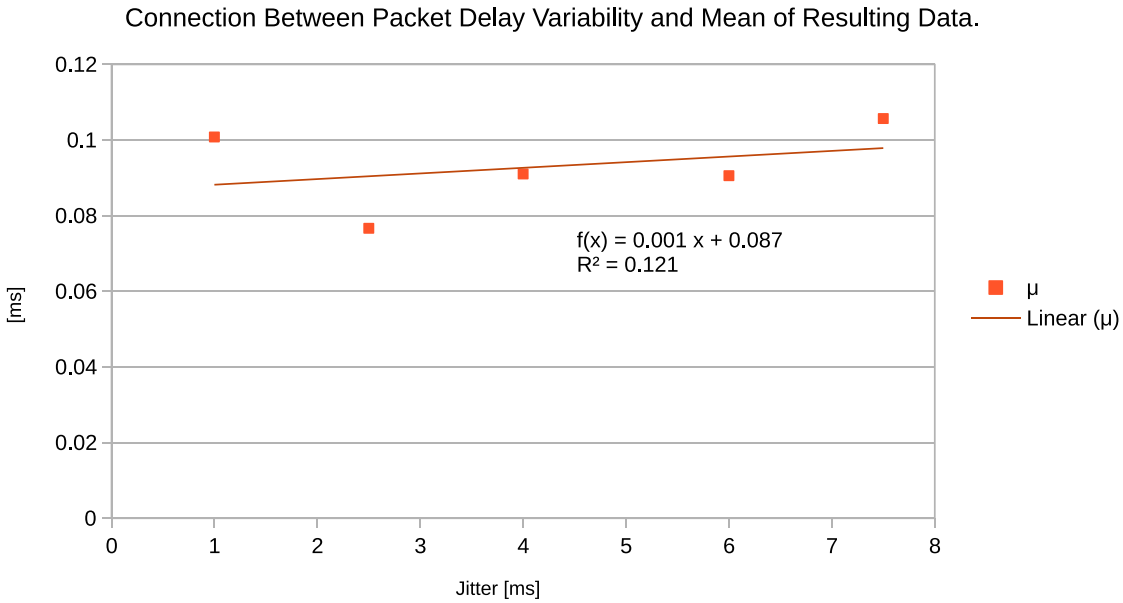


Figure 6.13: Measured mean at different packet delay jitter using RAMSES with fitted linear function.

Short-Term Performance

Figure 6.10 shows values for mean and standard deviation of the short-term measurements at different packet delay jitter values with fitted linear functions. The resulting short-term function for the standard deviation (in milliseconds) is $f(x) = 0.208x + 0.492$ with an R^2 of 0.960, the function (in milliseconds) for the mean is $f(x) = -0.041x + 0.0236$ with $R^2 = 0.759$.

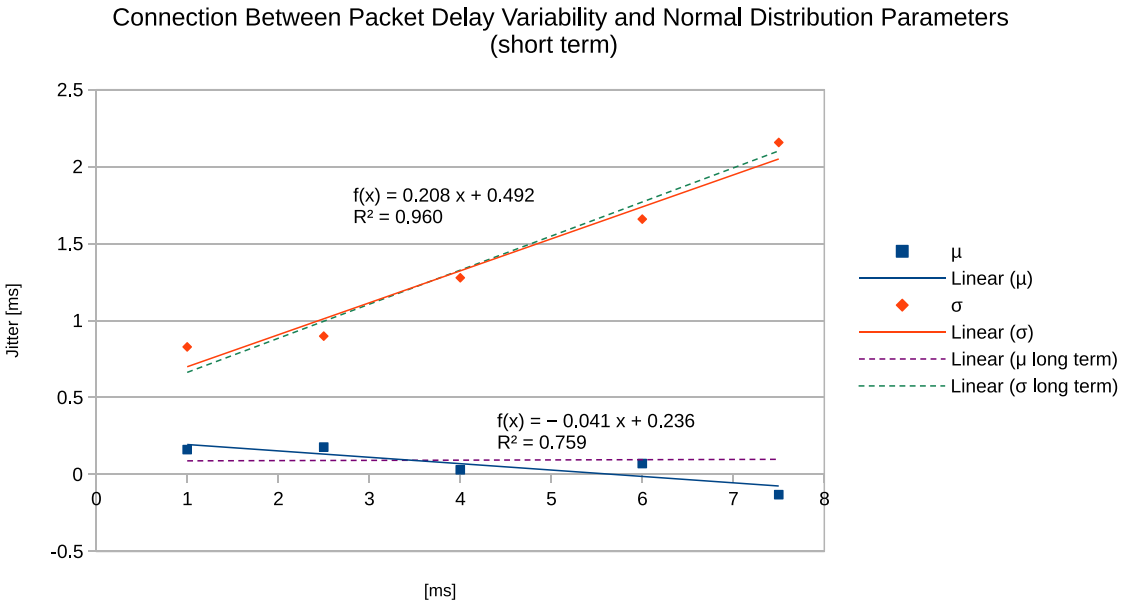


Figure 6.14: Connection between packet delay jitter and normal distribution parameters of the short-term results obtained using RAMSES.

6.2.2 Variable Packet Loss

RAMSES with packet loss is evaluated analogous to Baseline with packet loss to enable comparability. Multiple test bench runs were conducted, each one with a different percentage of packet loss in the system. Figure 6.15 shows the boxplots of the test bench results at different percentages of packet loss. Table 6.6 lists the quartiles associated with this data.

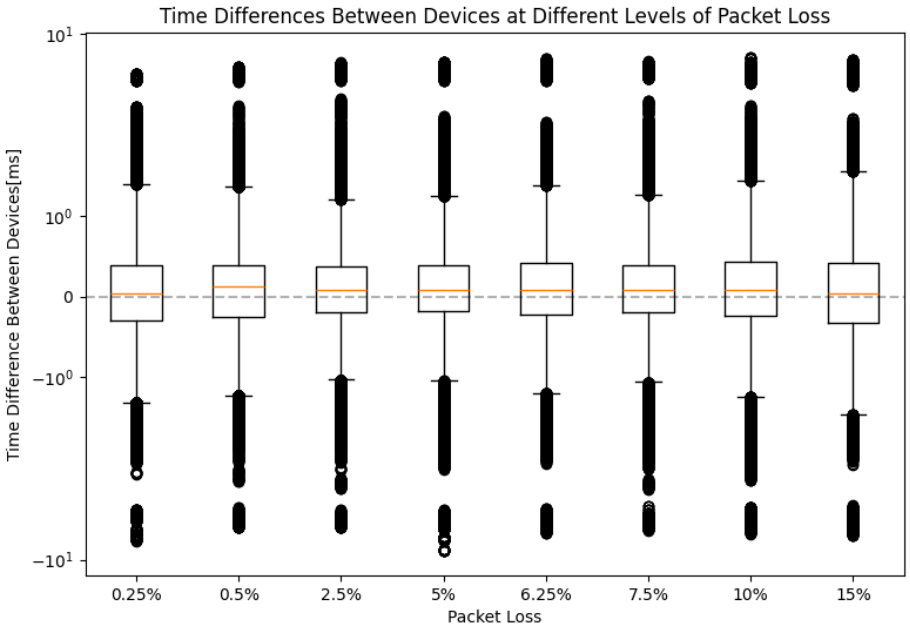


Figure 6.15: Boxplot of measured time differences between devices at different packet loss percentages using RAMSES.

.

Packet Loss	Min [ms]	Q1 [ms]	Q2 [ms]	Q3 [ms]	Max [ms]
0.25%	-7.42	-0.29	0.04	0.38	5.31
0.50%	-6.01	-0.25	0.13	0.40	5.99
2.50%	-5.9	-0.19	0.08	0.37	6.43
5.00%	-8.67	-0.18	0.08	0.39	6.44
6.25%	-6.56	-0.23	0.09	0.42	6.76
7.50%	-6.28	-0.19	0.08	0.39	6.45
10.00%	-6.63	-0.24	0.09	0.43	6.88
15.00%	-6.83	-0.33	0.04	0.42	6.68

Table 6.6: Minimum, quartiles 1-3 and maximum of the measured time differences between devices at different packet loss percentages using RAMSES.

In line with the baseline evaluation in subsection 6.1.2, datapoints of 50ms or more are classified as extreme outliers, removed, and the remaining datapoints are classified as outliers if they are more than 1.5 times the interquartile range greater than the third quartile or more than 1.5*IQR smaller than the first quartile. The results are listed in Table 6.7 and show that the number and percentage of outliers decrease with increasing packet loss percentage. To illustrate the relation between packet loss percentage and

the number of outliers, the values are plotted in Figure 6.16 together with the fitted second-degree polynomial function $f(x) = 45.44x^2 - 129699x + 11449.93$ ($R^2 = 0.93$).

Packet Loss	Extreme Outlier (>50 ms)	Outlier (1.5*IQR)	Total	Percentage
0.25%	0	9550	9550	6.16%
0.50%	0	11684	11684	7.53%
2.50%	0	9741	9741	6.28%
5.00%	0	6662	6662	4.3%
6.25%	0	4103	4103	2.65%
7.50%	0	4019	4019	2.59%
10.00%	0	3075	3075	1.98%
15.00%	0	2340	2340	1.51%

Table 6.7: Number of outliers at different packet loss percentages using RAMSES, determined by first removing extreme outliers (>50ms) and using the 1.5*IQR method to determine the number of outliers in the remaining data.

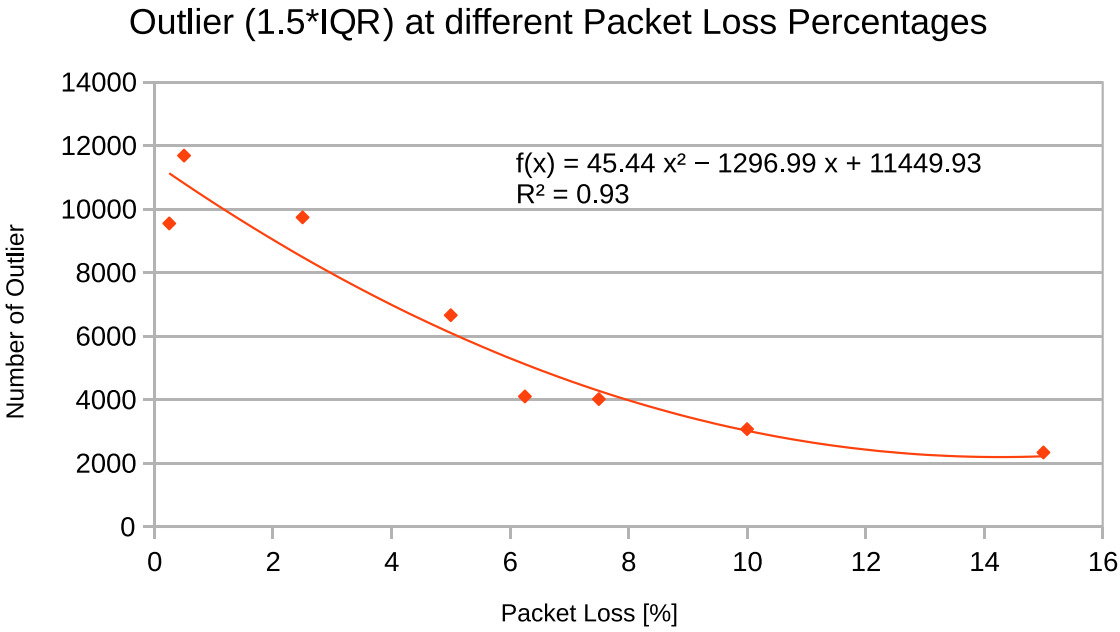


Figure 6.16: Number of outliers at different packet level percentages using RAMSES with fitted polynomial function of degree two.

In Figure 6.17 the quartiles 1, 2, and 3 at the different percentages of packet loss are plotted with linear functions fitted to them. The resulting R^2 values of 0.56, 0.14, and 0.09 indicate that the functions poorly fit the data points. This is an indicator that there is no linear relationship between the quartiles and the packet loss percentage. The data is interpreted in such a way that the quartiles are rather constant with variation in packet loss and the different quartile values are attributed to the fact that only a limited

amount of data was collected.

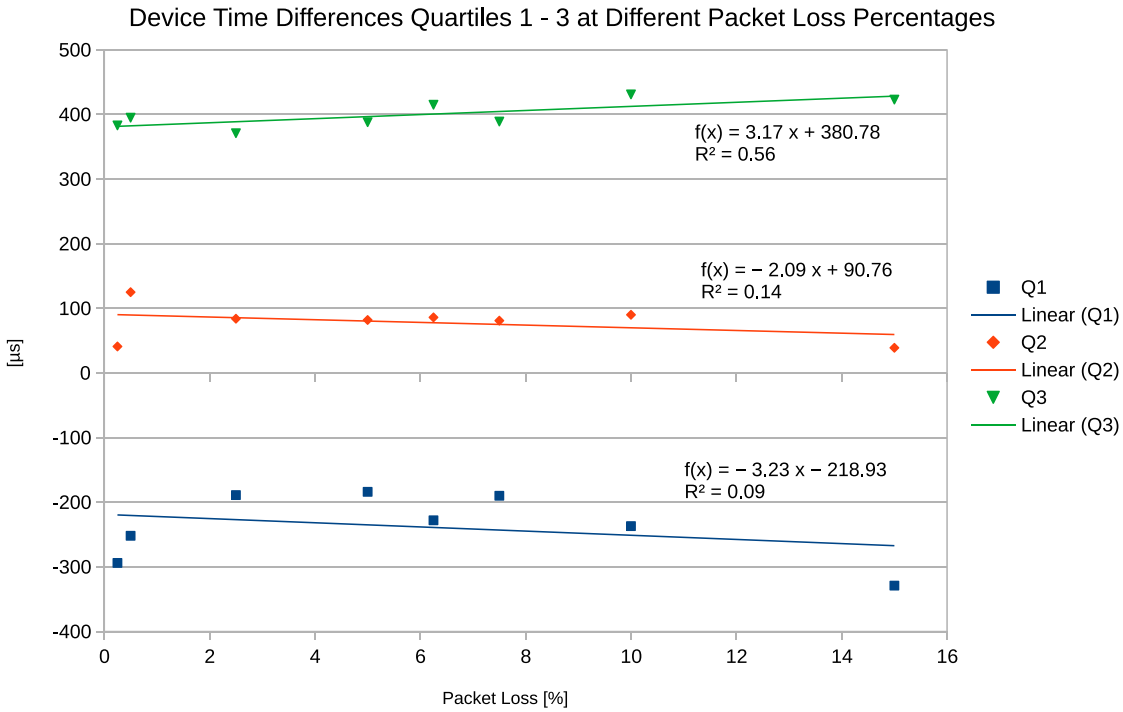


Figure 6.17: Measured quartiles different packet loss percentages using RAMSES with fitted linear functions.

All outliers were removed from the data and the results for the different packet loss percentages are displayed as KDE plot in Figure 6.18. Table 6.8 lists the mean and standard deviation of our measurement results after outliers were removed. Finally, mean and standard deviation at different percentages of packet loss are plotted together with fitted linear functions in Figure 6.19. Again, the R^2 values are rather small with 0.375 for the standard deviation and 0.026 for the mean. This suggests, that both mean and standard deviation are not linearly dependent on the packet loss percentage. We argue that these values are rather constant and that the variations in mean and standard deviation values for different packet loss percentages originate from the fact that only a limited amount of data was collected and thus noise is not sufficiently removed by the statistics.

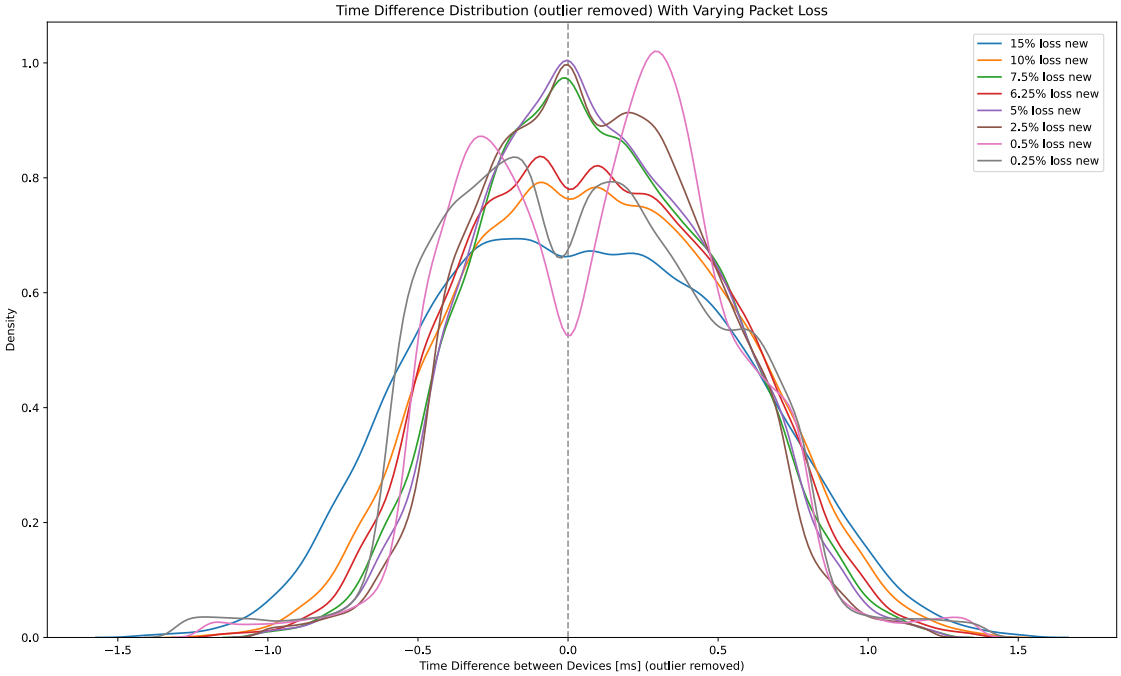


Figure 6.18: Distribution of relative time differences between devices at different packet loss percentages using RAMSES with outliers removed.

Packet Loss [%]	μ [μs]	σ [μs]
0.25	50.845	438.462
0.5	93.964	412.079
2.5	95.082	367.481
5	103.1	379.995
6.25	97.243	419.948
7.5	100.758	388.406
10	100.749	443.794
15	50.466	496.185

Table 6.8: Normal distribution parameters of measurement data obtained using RAMSES at different percentages of packet loss when outliers are removed. The packet loss percentages match the ones used for the Baseline approach. Those were selected to cover both a large range of percentages and different step sizes between percentages. Small step sizes at lower percentages should capture the effects of small changes while larger step sizes at higher percentages should cover general trends in the synchronization performance.

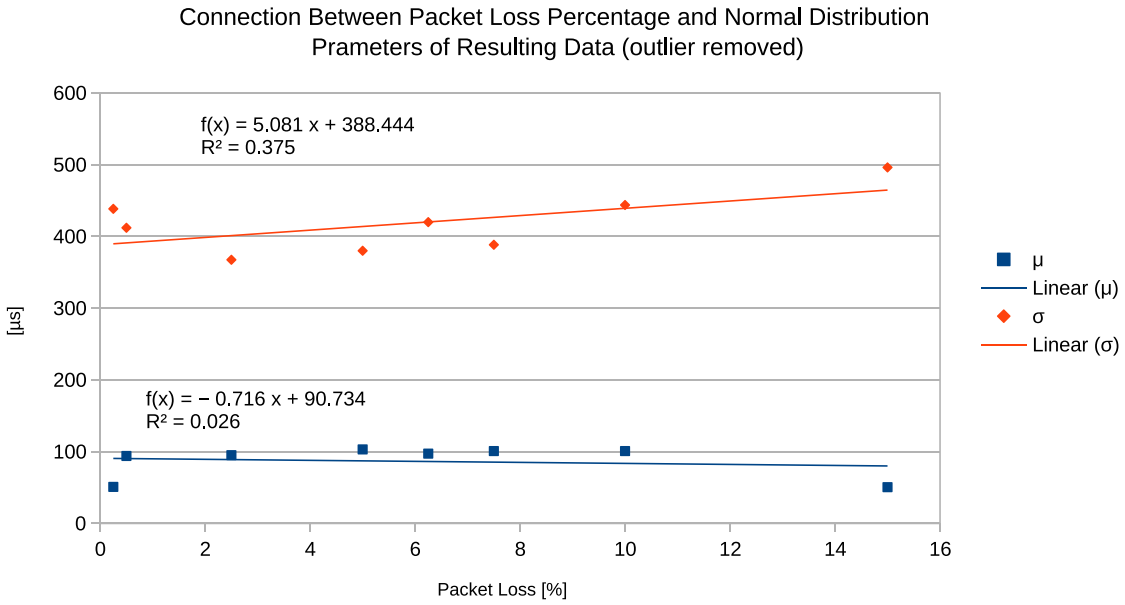


Figure 6.19: Measured mean and standard deviation with fitted linear functions after removal of outliers from measurement data obtained at different packet loss percentages using RAMSES.

Short-Term Performance

Figure 6.20 shows values for mean and standard deviation of the short-term measurements at different packet loss percentages with fitted linear functions. The resulting short-term function for the standard deviation (in microseconds) is $f(x) = 10.24x + 343.02$ with an R^2 of 0.84, the function (in microseconds) for the mean is $f(x) = -3.88x + 118.29$ with $R^2 = 0.35$.

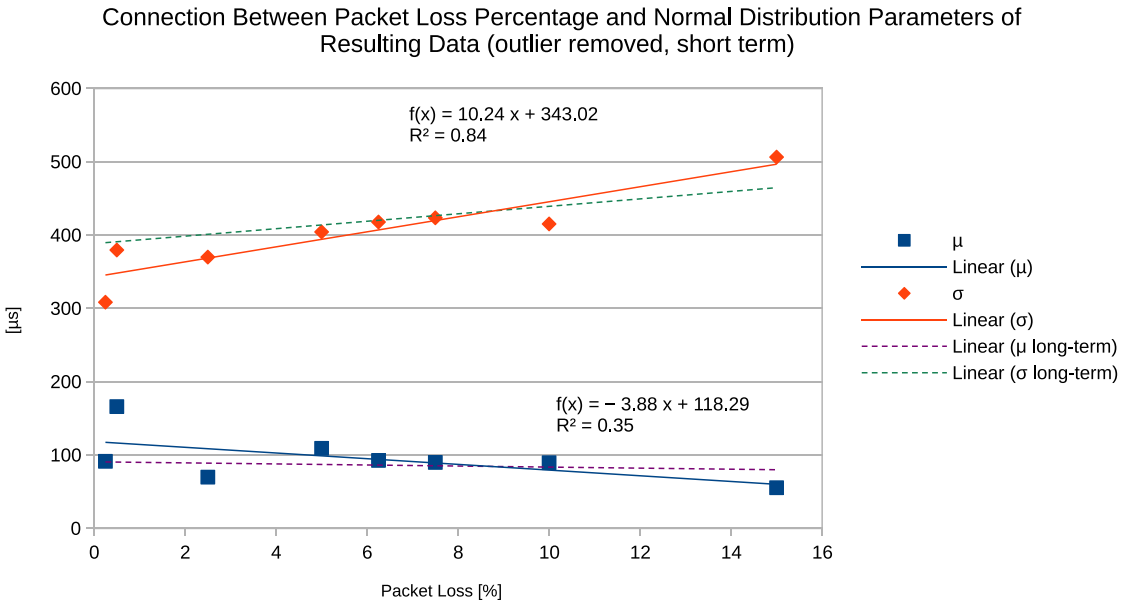


Figure 6.20: Connection between packet loss percentages and normal distribution parameters of the short-term measurement results obtained using RAMSES.

6.2.3 RAMSES Evaluation Summary

When facing asymmetric packet loss, the RAMSES approach exhibits a small but linear relationship between the packet delay jitter and the spread of synchronization results. The standard deviation of the results in milliseconds is described by the function $f(x) = 0.221x + 0.441$ ($R^2 = 0.995$), where x is the standard deviation (jitter), of the random delay applied by our test bench.

For packet delay, RAMSES is not affected by extreme outlier (>50ms) and shows a quadratically decreasing number of 1.5*IQR outlier with increasing packet delay. The characterizing curve is $45.44x^2 - 1296.99x + 11449.93$ with $R^2 = 0.93$.
Our measurements also show that RAMSES performs slightly worse with small packet loss when compared to the performance with medium packet loss.

6.3 Combination of Delay Jitter and Packet Loss

The previous sections evaluate the performance of Baseline and RAMSES when facing asymmetric packet delay and packet loss separately. In practice, however, it will be very unlikely to experience only one of them without the other. To get a better understanding of the performance of Baseline and RAMSES with a combination of packet loss and delay, two test scenarios called Combined High and Combined Medium were defined, in which a high (respectively medium) number of packet loss and asymmetric delay are simulated (Table 6.9).

	Combined High	**Combined Medium**
Packet Loss To Time Server	15%	5%
Packet Loss From Time Server	15%	5%
Delay Jitter To Time Server	0 ms	0 ms
Delay Jitter From Time Server	7.5 ms	4 ms

Table 6.9: Packet loss and delay values for the two scenarios Combined High and Medium.

Figure 6.21 illustrates the distribution of time differences between devices for both approaches and combined scenarios as KDE plot.

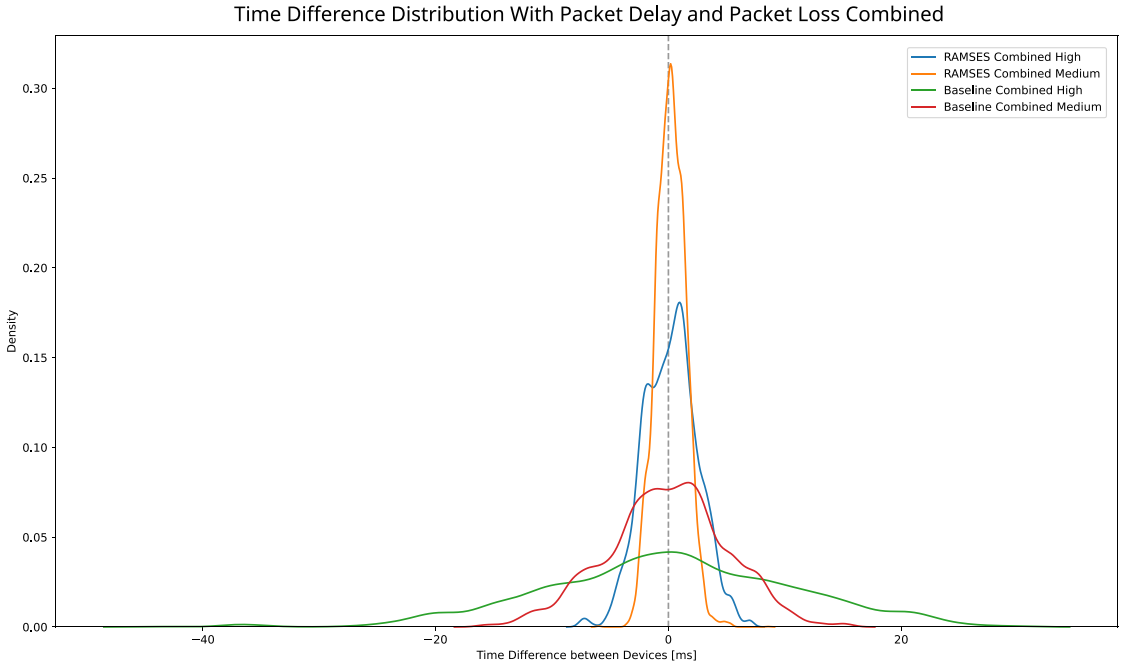


Figure 6.21: Distribution of the relative time differences for two different combinations, High and Medium, of both, packet loss and asymmetric delay using Baseline and RAMSES.

	Combined High		Combined Medium	
	µ [ms]	$\sigma[ms]$	$\mu[ms]$	$\sigma[ms]$
Baseline	0.12	10.81	-0.12	5.13
RAMSES	0.20	2.3	0.20	1.28

Table 6.10: Normal distribution parameters of the test bench results for Baseline and RAMSES with Combined High and Combined Medium.

6.4 Time Synchronization Artifacts

The test bench results contain artifacts in the form of **jumps in the time difference between devices**. These jumps occur around the (re)synchronization of the devices with the time server, when the pulse from the signal source arrives after some but before all devices synchronized with the time server. Another prerequisite for these jumps is that the local time is adjusted due to the time synchronization. In the case of zero clock skew and perfect synchronization without any offset error, these jumps would not happen.

To illustrate how and why these jumps occur, three idealized scenarios are used, in which clock skew only, synchronization error only, and a combination of the two are considered. For all scenarios, large time spans between synchronizations and short time spans between signal source pulses are assumed, so that when plotting only a few time differences from consecutive measurement values without a resynchronization occurring between them, the clock skew is barely perceptible or not perceptible at all.

No Synchronization Errors, Different Clock Skews

In this scenario, perfect, offset error-free time synchronization of the devices is assumed. However, each device has a different clock skew. This results in the devices being perfectly synchronized just after the synchronization but off by a couple of milliseconds just before the resynchronization (Table 6.11).

	offset after sync.	offset before sync.
D1	0ms	3ms
D2	0ms	5ms
D3	0ms	6ms

Table 6.11: Offset values before and after synchronization due to clock skew per device

Figure 6.22 displays the jump at time `t`, that is caused because `D1` synchronizes its time before `t`, between `t-1` and `t`, and `D2` and `D3` synchronize their time after `t`, between `t` and `t+1`.

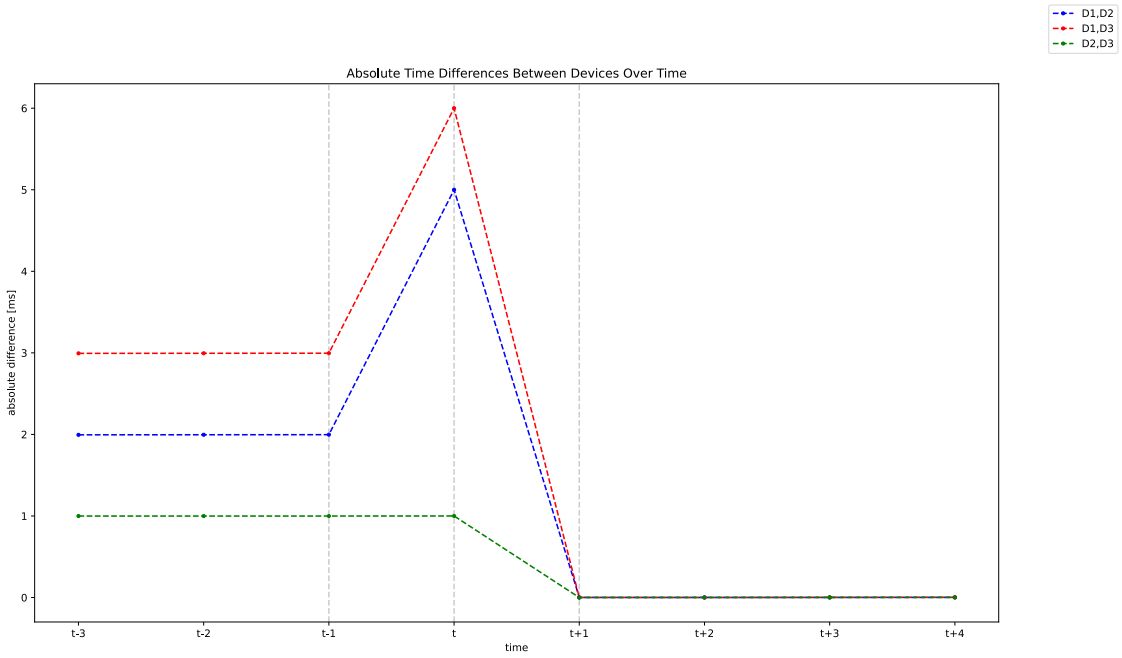


Figure 6.22: Jump caused by different clock skews.

Different Synchronization Errors, No Clock Skews

In this scenario, no clock skew between synchronizations but different time offset errors resulting from the synchronization processes are assumed. Two synchronizations are considered, the first has already happened and the values around the second synchronization are examined. The assumed values can be found in Table 6.12.

	first synchronization	synchronization (`t`)
D1	1ms	2ms
D2	-1ms	2.5ms
D3	2ms	-0.5ms

Table 6.12: Assumed offset errors occurring when synchronizing.

Figure 6.23 shows the jumps at time `t`, that is caused by D2 synchronizing before `t`, between `t-1` and `t`, while `D1` and `D3` synchronize their time after `t`, between `t` and `t+1`. Before synchronizing, the devices have a constant time offset from the first synchronization, after synchronizing the offset changes to the offsets caused by that second synchronization.

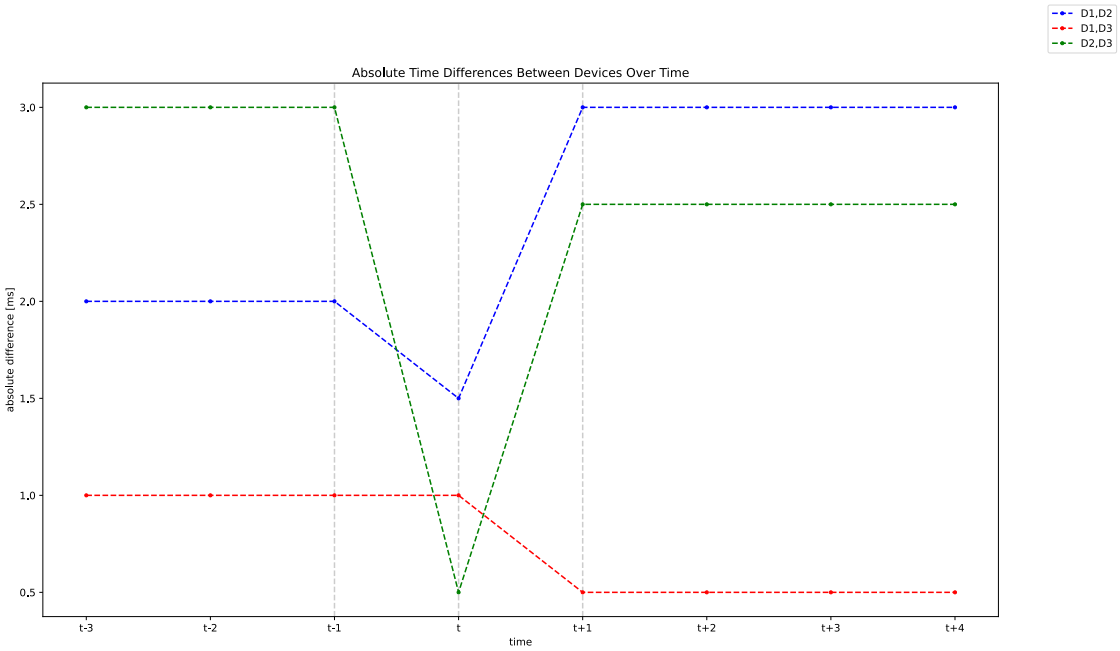


Figure 6.23: Jump caused by different offset errors at synchronization.

Different Synchronization Errors, Different Clock Skews

In practice, jumps will originate from a combination of synchronization errors and different clock skews. Again it is assumed that the first synchronization has already happened and the values around the second synchronization at time `t` are being examined. The assumed synchronization errors and clock skews are listed in Table 6.13. The skew is assumed to be zero right after synchronization and builds linearly over the period between the synchronizations until it reaches the maximum value just before synchronization happens.

	first sync.	second sync. (`t`)	max. skew
D1	1ms	1.5ms	2ms
D2	2ms	2.5ms	6ms
D3	-3ms	-2ms	7ms

Table 6.13: Assumed skew and offset errors occurring when synchronizing.

Unlike the example in the previous section, in Figure 6.24, before `t` not only the absolute difference in offset synchronization errors between the devices can be seen but also the effects of the clock skews. For example, the difference between the time offsets that originate from synchronization errors would be 1ms for `D1,D2`. However, due to the clock skew, `D1,D2` are at around 5ms. This is because just before synchronization, `D1` is off by 3ms (1ms + 2ms) and `D2` is off by 8ms (2ms + 6ms).
The jump happens because `D1` synchronizes before `t`, between `t-1` and `t`, while `D2` and

D3 synchronize their time after t, between t and t+1.
After all synchronizations happened, at t+1, the absolute differences are only based on the synchronization errors from the second synchronization. This is because the skew has yet to accumulate until it has an influence that is visible in the figure.

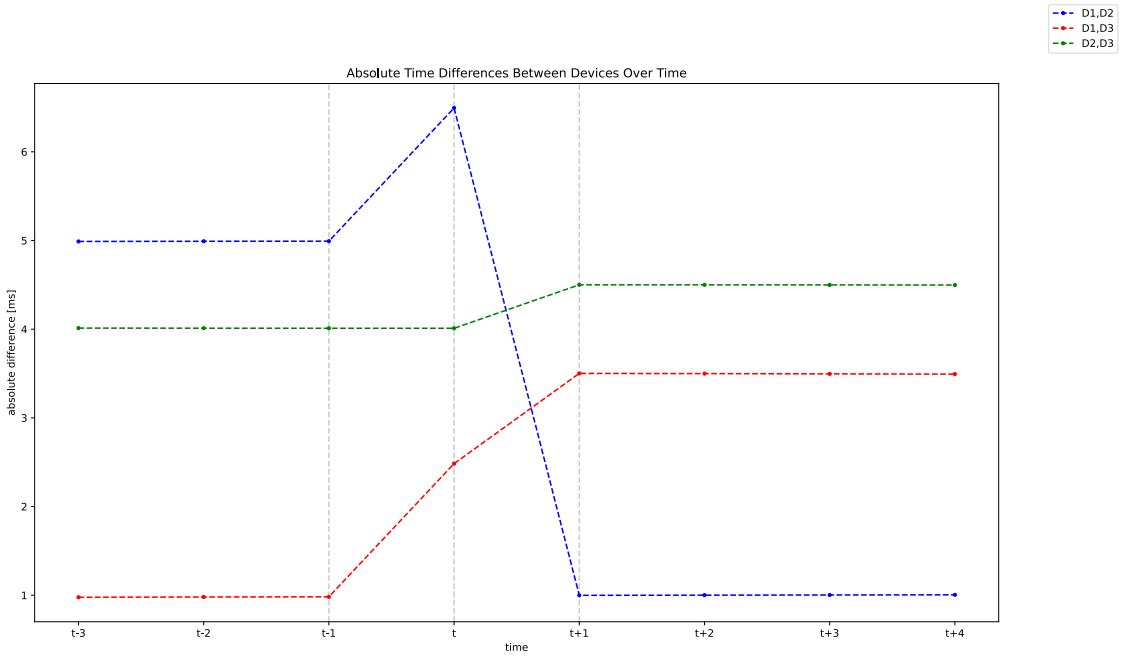
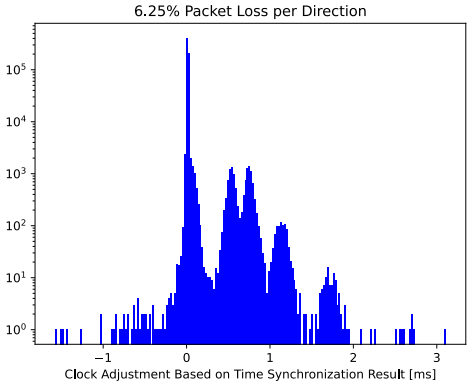


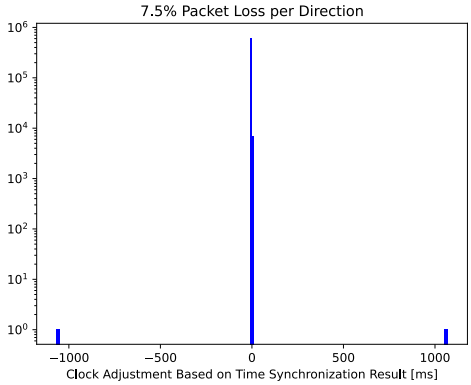
Figure 6.24: Jump caused by different offset errors at synchronization in combination with clock skews.

6.5 Extreme Outliers in Baseline Results

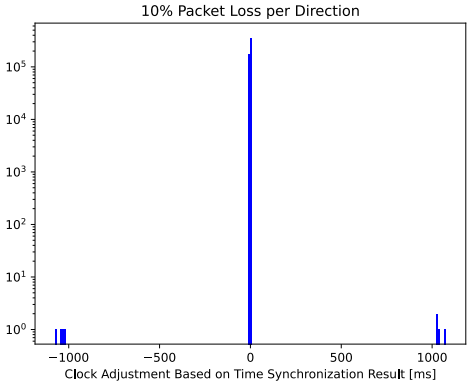
In subsection 6.1.2 extreme outliers appearing with packet loss percentages of 7.5% and more were reported. In this section, the origins of these outliers are further investigated. As can be seen in Figure 6.25, the outliers show a suspicious accumulation around one and two seconds. For all outliers, the same pattern can be observed. Outliers always appear in pairs (see Figure 6.27). The first outlier is due to the clock being set back about approximately one (or two respectively) second because of the synchronization with the time server. The next outlier appears at the next synchronization with the time server and advances the clock by approximately one (or two respectively) second. All of the outliers in our data follow the pattern of first retreating, and then advancing the time. In our data, a correlation between the number of outliers and the amount of simulated packet loss in the test setup can be observed (Figure 6.28).
Such outliers are only observed when simulating packet loss of 7.5% and more on both



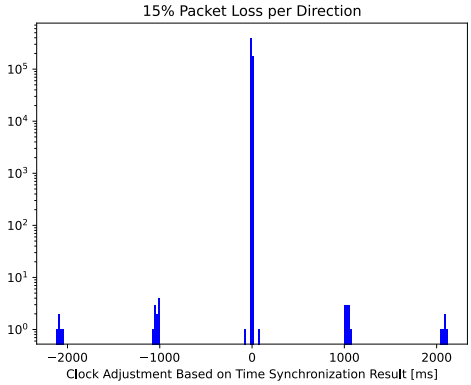
(a) 6.25% packet loss per direction. No outliers.



(b) 7.5% packet loss per direction. Outliers at about 1 second.



(c) 10% packet loss per direction. More outliers at about 1 second.



(d) 15% packet loss per direction. Even more outliers at about 1 second, outliers at about 2 seconds too.

Figure 6.25: Histograms of time adjustment values that are the result of the synchronization process with different packet loss percentages using the Baseline approach.

paths, to and from the time server. Extreme outliers were neither observed with 6.25% packet loss or below nor when simulating different asymmetric delays (Table 6.14). Outliers appear on all devices in our setup (Figure 6.29) and are independent of the used time server software (tested with chrony and ntpd) and used switch hardware (tested with Netgear and TP-Link).

Intuitively, when simulating packet loss only in our test setup, it is assumed that packets are either transmitted without great delays or lost completely. However, when inspecting the time synchronization packets, it was discovered that the outliers occur because sometimes packets take one or two seconds on their way from the time server to the microcontroller. The on-wire protocol packet contains three timestamps. T1 is the transmission time of the client, the microcontroller in our scenario. T2 is the timestamp of when the time server received the packet and T3 is the timestamp at which the time server sent out the response packet. The fourth timestamp T4 is the timestamp at which the microcontroller has received the response packet. This timestamp is not part of the packet anymore since it is never transmitted over the network. The time difference of one second (two respectively) is between T3 and T4, while T1 to T3 are all within a couple of hundred microseconds.

After thoroughly investigating this issue, the culprit was identified to be the Address Resolution Protocol (ARP) in combination with packet loss. ARP is used in networks to determine the Media Access Control (MAC) addresses associated with IP addresses. MAC addresses are needed to ensure that packets are sent to the proper recipient in the local network. Once a MAC address for an IP address is known, it's typically stored in a cache. However, because IP addresses are generally not permanently linked to devices but can be re-assigned to other devices, these caches are typically quite ephemeral.

Take for example the situation where a Dynamic Host Configuration Protocol (DHCP) server in a network assigned device D_1 the IP IP_a. Every other participant in this network that wants to send something to IP_a needs to first determine the MAC address $MAC(IP_a)$. This is done via ARP and the result is cached so that the ARP request result can be reused for further communication. Now, if D_1 exits the network, D_2 joins, and the DHCP server re-assigns IP_a to D_2, everyone that wants to send something to IP_a would need to send packets to the MAC address of D_2. As long as the ARP cache contains the MAC address of D_1 for IP_a, however, packets to IP_a are sent to the MAC address of D_1, and D_2 never receives anything. Because of that, ARP cache entries are typically short-lived (think tens of seconds). Once the cache entry has expired, a new ARP request is sent to the network to ensure the actuality of the entries.

Network traffic was analyzed to observe the behavior of the ARP implementation of the used time server. Figure 6.26 is a Wireshark screenshot that shows the packets around the time synchronization that resulted in an extreme outlier. Lines 1778, 1781, and 1782 are ARP requests sent by our time server directly to the MAC address of the device in question. These requests are about one second apart each. What cannot be seen are responses to these ARP requests, and that in between these ARP requests, NTP packets are sent between time server and client. NTP packets always appear in pairs. First the client (`192.168.33.123`) sends a packet to the server (`192.168.33.33`),

No.	Time	Source	Destination	Protocol	Length	Info
1774	22:00:20.824134	192.168.33.123	192.168.33.33	NTP	90	NTP Version 4, client
1775	22:00:20.824293	192.168.33.33	192.168.33.123	NTP	90	NTP Version 4, server
1776	22:00:22.826179	192.168.33.123	192.168.33.33	NTP	90	NTP Version 4, client
1777	22:00:22.826464	192.168.33.33	192.168.33.123	NTP	90	NTP Version 4, server
1778	22:00:23.841159	[redacted]:ea	[redacted]:5d	ARP	42	Who has 192.168.33.123? Tell 192.168.33.33
1779	22:00:24.828250	192.168.33.123	192.168.33.33	NTP	90	NTP Version 4, client
1780	22:00:24.828436	192.168.33.33	192.168.33.123	NTP	90	NTP Version 4, server
1781	22:00:24.881157	[redacted]:ea	[redacted]:5d	ARP	42	Who has 192.168.33.123? Tell 192.168.33.33
1782	22:00:25.921154	[redacted]:ea	[redacted]:5d	ARP	42	Who has 192.168.33.123? Tell 192.168.33.33
1783	22:00:26.830980	192.168.33.123	192.168.33.33	NTP	90	NTP Version 4, client
1784	22:00:26.831201	192.168.33.33	192.168.33.123	NTP	90	NTP Version 4, server
1785	22:00:30.849716	192.168.33.123	192.168.33.33	NTP	90	NTP Version 4, client
1786	22:00:30.850056	[redacted]:ea	Broadcast	ARP	42	Who has 192.168.33.123? Tell 192.168.33.33
1787	22:00:31.921157	[redacted]:ea	Broadcast	ARP	42	Who has 192.168.33.123? Tell 192.168.33.33
1788	22:00:31.922698	[redacted]:5d	[redacted]:ea	ARP	60	192.168.33.123 is at [redacted]:5d
1789	22:00:31.922726	192.168.33.33	192.168.33.123	NTP	90	NTP Version 4, server
1790	22:00:33.924753	192.168.33.123	192.168.33.33	NTP	90	NTP Version 4, client
1791	22:00:33.925071	192.168.33.33	192.168.33.123	NTP	90	NTP Version 4, server
1792	22:00:35.926846	192.168.33.123	192.168.33.33	NTP	90	NTP Version 4, client
1793	22:00:35.927036	192.168.33.33	192.168.33.123	NTP	90	NTP Version 4, server
1794	22:00:37.929125	192.168.33.123	192.168.33.33	NTP	90	NTP Version 4, client
1795	22:00:37.929386	192.168.33.33	192.168.33.123	NTP	90	NTP Version 4, server

Figure 6.26: Wireshark screenshot that shows the delay of NTP packet due to missing ARP responses.

which nearly immediately sends an NTP packet back. However, in line 1785 there is an NTP Packet from client to server and the next NTP Packet from server to client is found about 1 second later in line 1789. In between ARP requests happened. The first one (line 1786) is as before but as broadcast, the second one (line 1787) a second later, and this time followed by the expected response in line 1788.
The pattern can be observed for all extreme outliers in our data. It shows that the ARP implementation of the time server tries to refresh its APR cache entry by asking the last known IP holder directly if the mapping is still correct and there seems to be a one-second timeout for unresponded requests. Until the timeout for the third attempt has expired, the ARP cache entry seems to still be used for communication. After three failures to receive an answer from the last known IP holder, the cached value is not used anymore and the whole network (broadcast) is asked for the MAC address for the given IP. These broadcast requests have a one-second timeout too. After an answer to the ARP request is received, the packets that were delayed due to the missing information about the recipient's MAC address are sent out.
So for an extreme outlier of one second to occur, four subsequent ARP requests must get lost: three direct requests to the previously known holder and one broadcast request. For an extreme outlier of two seconds, an additional broadcast request must get lost. These requirements also explain why extreme outliers were only observed with 7.5% packet loss and more. A high probability of packet loss is needed for such a scenario to occur because if just one of the aforementioned requests (or the associated answer) is not lost, everything works without any problems.

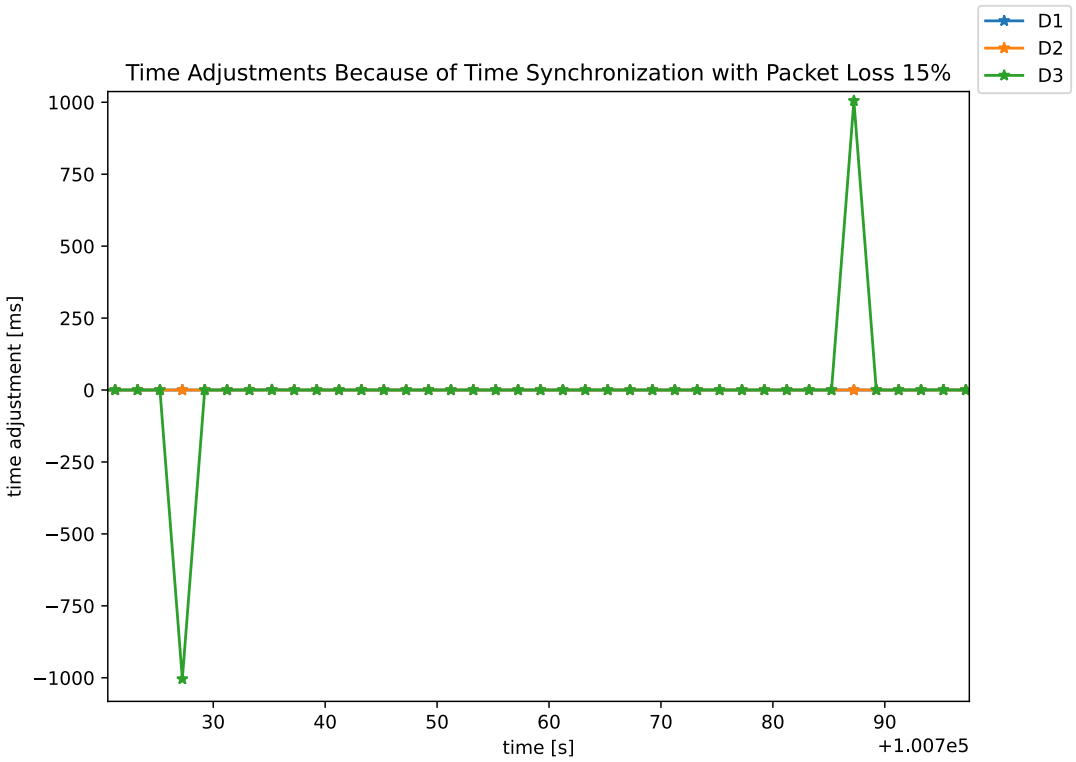


Figure 6.27: Example of a typical outlier pair observed in the data.

Packet Loss per Direction	Number of Outlier Pairs ∼1s	Number of Outlier Pairs ∼2s
0.25%	-	-
0.50%	-	-
2.50%	-	-
5.00%	-	-
6.25%	-	-
7.50%	2	-
10.0%	4	-
15.0%	10	5

Table 6.14: Number of 1s and 2s outlier pairs at different percentages of packet loss.

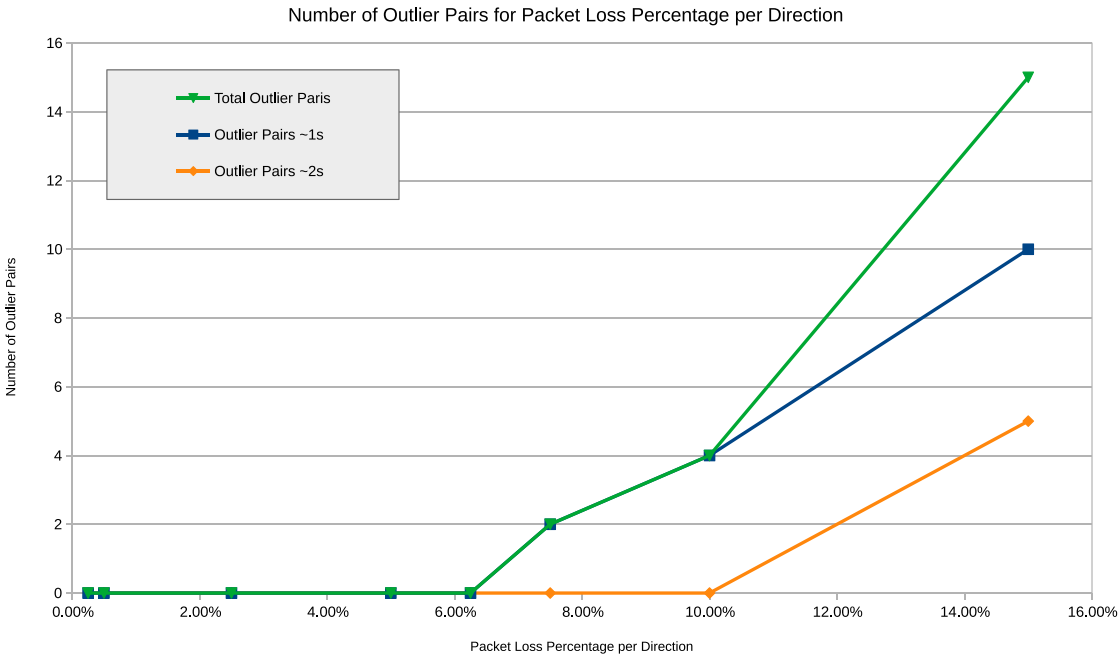


Figure 6.28: Number of outliers at different percentages of packet loss in our setup.

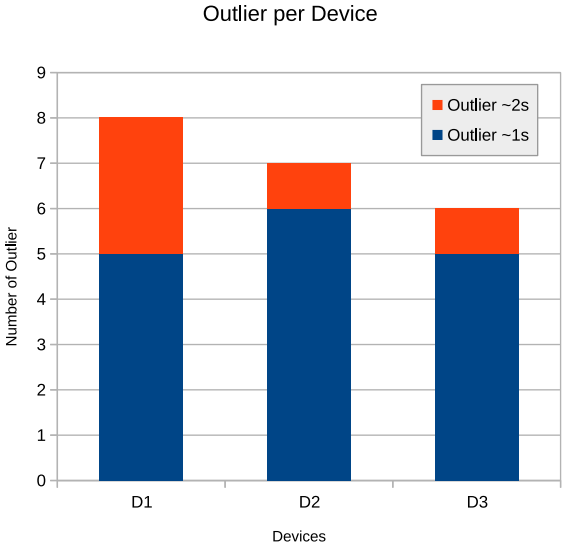


Figure 6.29: Number of outliers per device in our setup.

6.6 Distribution of Test Bench Results

In the test bench, time measurements from the microcontrollers are collected every two seconds but the microcontrollers only synchronize their time once every 60 seconds. Consequently, the raw data resulting from the test bench does not represent the distribution of the synchronization results. There is one synchronization result about every 30 entries (varies because of delays) and after one synchronization result come about 30 values that are dependent on both the synchronization result and the clock drift of the microcontroller. This results in a distribution that is not trivially classifiable. For the evaluation of synchronization approaches the relative time difference between the microcontrollers is used. The KDE plots of those measurements shown in Figure 6.30 and Figure 6.31 all exhibit a bell-shaped curve, visually resembling a normal distribution. To simplify the comparison between approaches, a normal distribution of the measurements is assumed, and test bench results are compared based on mean and standard deviation.

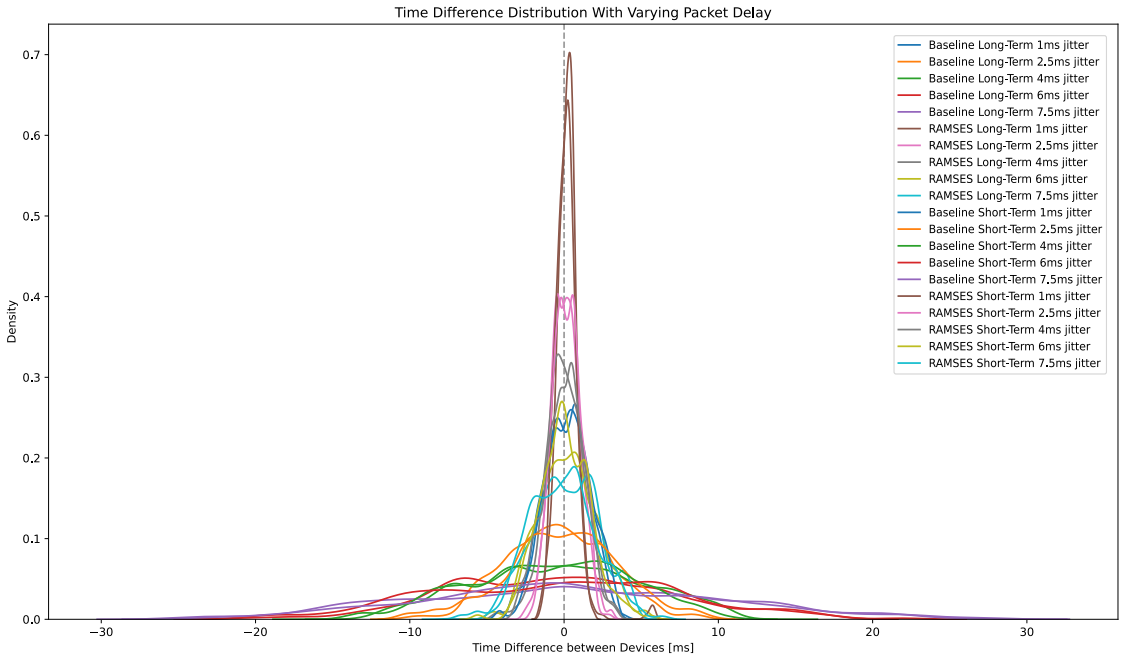


Figure 6.30: Distribution of relative time differences between the devices in long-term and short-term asymmetric packet delay measurements for Baseline and RAMSES.

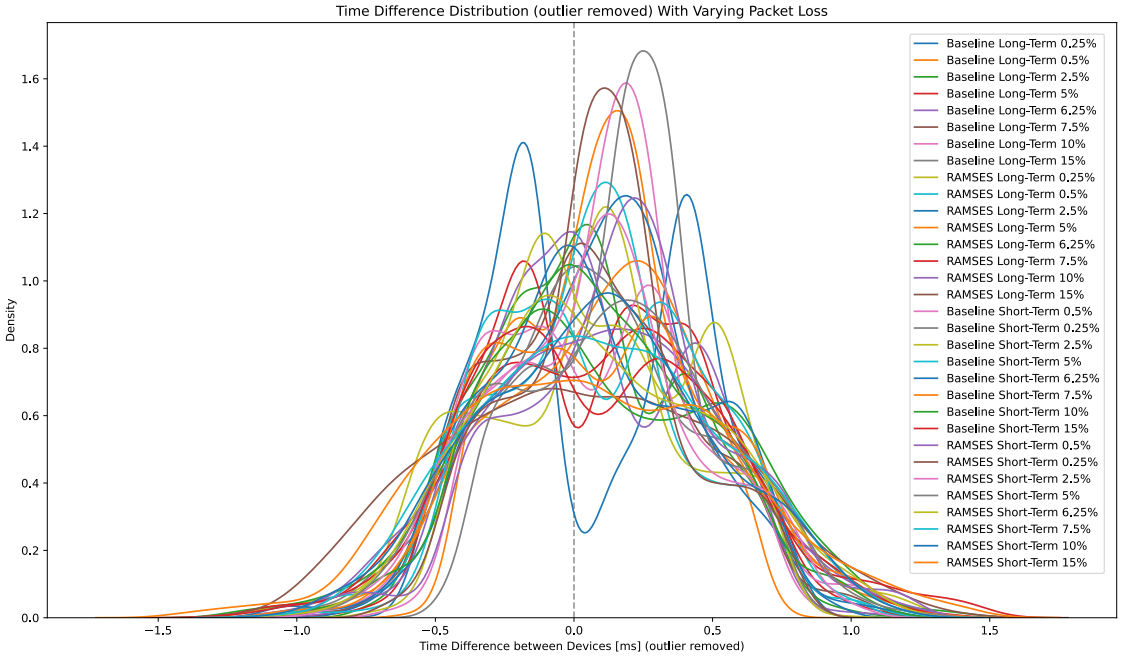


Figure 6.31: Distribution of relative time differences between the devices in long-term and short-term packet loss measurements for Baseline and RAMSES.

CHAPTER 7

Results and Discussion

Multiple measurements were performed with two different synchronization approaches, Baseline and RAMSES. Baseline is the default way to synchronize the time that comes with the manufacturer SDK for the microcontroller used in this thesis. In our case, this is the ESP-IDF that uses the SNTP functionality provided by the contained lwip stack. RAMSES is our own synchronization approach that is based on the (S)NTP on-wire protocol. To improve the synchronization results over the Baseline, in our approach, the on-wire protocol is executed multiple times, the results are filtered for outliers and then combined to determine a result that is less influenced by asymmetric packet delays and packet loss.
In this chapter, the measurement results are analyzed and it is discussed how RAMSES performs compared to the Baseline approach.

7.1 Asymmetric Packet Delay

To investigate the performance of the two approaches with different asymmetric packet delay, different jitter values were simulated on the path from the time server to the microcontrollers while the simulated delay on the path from the microcontrollers to the time server was kept constant.
In the evaluation, it was assumed that the sets of relative time differences are normally distributed (section 6.6), and mean and standard deviation were calculated for every measurement. Figure 7.1 shows the standard deviation of all measurements for both synchronization approaches. For both approaches, the standard deviation of the relative time differences increases with increasing jitter (the standard deviation of the distribution where the delay values come from). However, with our approach, the connection is not that strong. To quantify this difference, linear functions were fitted to the data points of both approaches. The resulting functions are $f(x) = 1.288x + 0.156$ with $R^2 = 0.998$ for

the Baseline approach and $f(x) = 0.221x + 0.441$ with $R^2 = 0.995$ for RAMSES. These functions show that when using RAMSES, increasing jitter has a lot less (about 82% less) influence on the standard deviation of the relative time differences than when using the Baseline approach. An ideal synchronization approach would perform equally well no matter the jitter and thus the fitted linear function would be of the form $f(x) = 0x + c$. With a factor of 0.221 for x, our approach is closer to an ideal approach than the Baseline with a factor of 1.288. Thus, for asymmetric packet delay, our approach performs better than the Baseline approach.

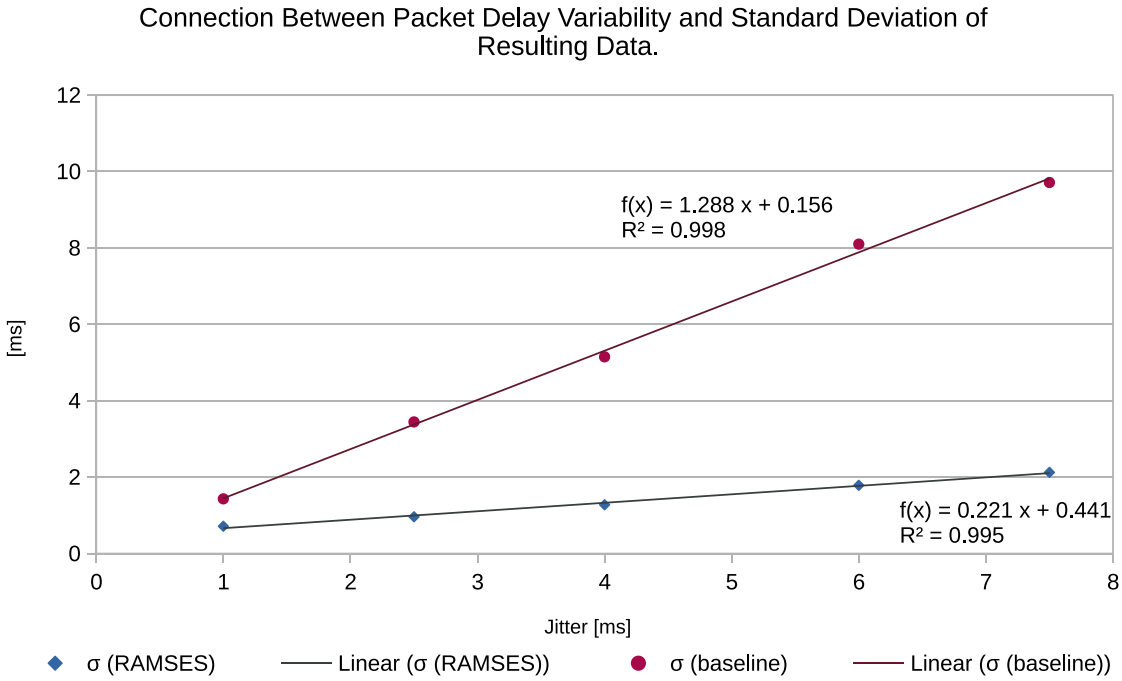


Figure 7.1: Connection between packet delay and standard deviation of the relative time differences between devices with Baseline and RAMSES.

Figure 7.2 shows the mean values of all measurements for both synchronization approaches. Recall that every measurement is a set of relative time differences between devices and the delay applied to the network packets follows a normal distribution with a defined mean and standard deviation (μ_d, σ_d) per measurement setting. Since the delay process is the same for all devices and the delay application is independent and identically distributed (i.i.d.), the delays applied to the packets from a single device come from (μ_d, σ_d) and it is expected that they follow the same normal distribution (μ_d, σ_d). So the mean of relative differences should be 0. Figure 7.2 shows that the mean does not reach zero in our measurements. However, rather small values of about 0.05 to 0.2 milliseconds can be seen. Fitted linear functions to the mean values of both approaches show rather small factors for x and quite low R^2 values. This indicates that the functions do not fit the data particularly well. So we argue that the differences in mean values between measurements are to be attributed to the limited number of collected data points. The used dataset

contains about 180000 samples. With one sample collected every two seconds and one time synchronization per minute, the dataset contains about 6000 time synchronizations.

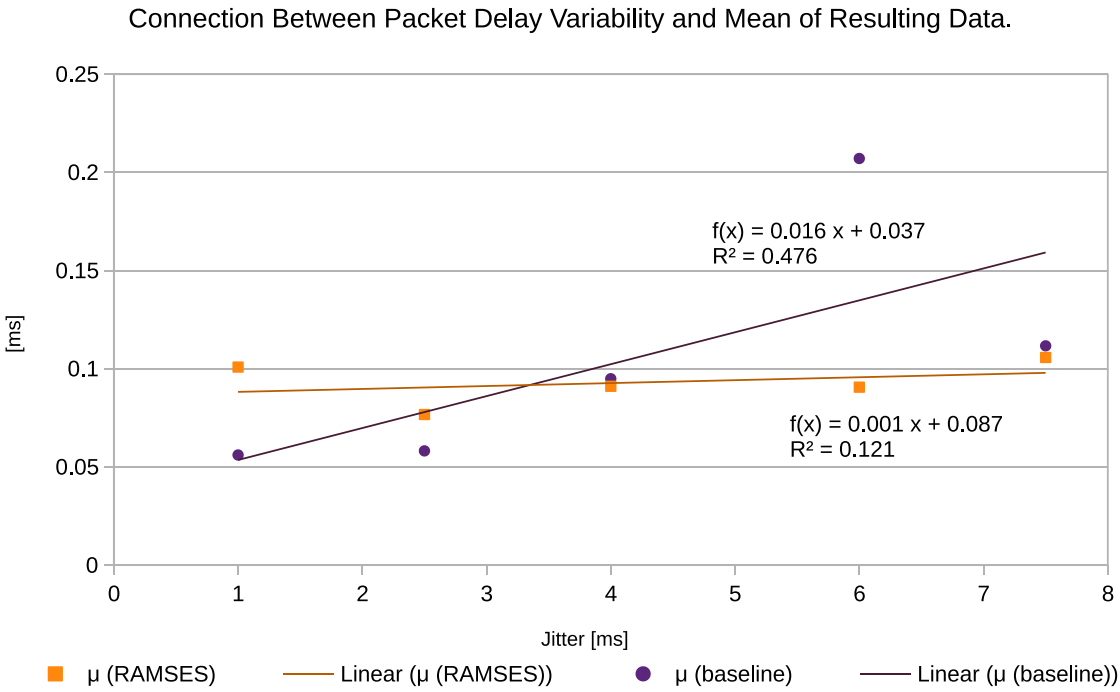


Figure 7.2: Connection between packet delay and mean relative time difference between devices with Baseline and RAMSES.

7.2 Packet Loss

To assess how Baseline and RAMSES perform when facing packet loss, different percentages of packet loss in both directions, to and from the time server were simulated. For example, when simulating 15% packet loss in our setup, about 15% of packets are lost on their way to the time server and about 15% are lost on the return path. The probability for a packet to reach the time server p_{in} is thus $1-0.15 = 0.85$. The probability for a response packet from the time server to reach the recipient $p_{out} = 1-0.15 = 0.85$. The probabilities are independent, so the probability for a time synchronization client to successfully get a response to a request to the time server $p_{success} = p_{in} \times p_{out} = 0.85 \times 0.85 = 0.7225$. This leaves us with a total loss probability $p_{loss} = 1 - p_{success} = 1 - 0.7225 = 0.2775 = 27.75\%$. For the evaluation of packet loss, the relative time difference between the devices from the test bench results was determined, outliers were removed, a normal distribution of the resulting data was assumed (section 6.6) and thus the two approaches can be compared by their mean and standard deviation values.
Figure 7.3 shows the standard deviation values for both approaches at different packet loss percentages determined in chapter 6. For the Baseline approach, a small but steady increase in the standard deviation with increasing packet loss can be seen. This is quantified

by the fitted linear function for the standard deviation in milliseconds $f(x) = 0.01x+0.413$. The R^2 of 0.967 is a strong indicator for a proper linear relationship between packet loss percentage and the standard deviation value for the Baseline approach. For the RAMSES approach, the standard deviation values are slightly higher but do not really seem to be directly related to the packet loss percentage. The R^2 value of 0.375 (fitted function $f(x) = 0.005x + 0.388$ for standard deviation in milliseconds) also indicates a lack of linear relationship between the two. About 155000 measurements were collected. In the test bench, a measurement is taken every two seconds, and time is synchronized every 60 seconds. This equals a little bit more than 5000 synchronizations. We attribute the variation in standard deviation at different packet loss percentages to the limited amount of data used for our evaluation and argue, that with more measurements, the standard deviations should become more constant, as is the case when looking at short-term measurements ($f(x) = 10.24x + 343.02$ standard deviation in µs) vs. long-term measurements ($f(x) = 5.081x + 388.44$ standard deviation in µs).

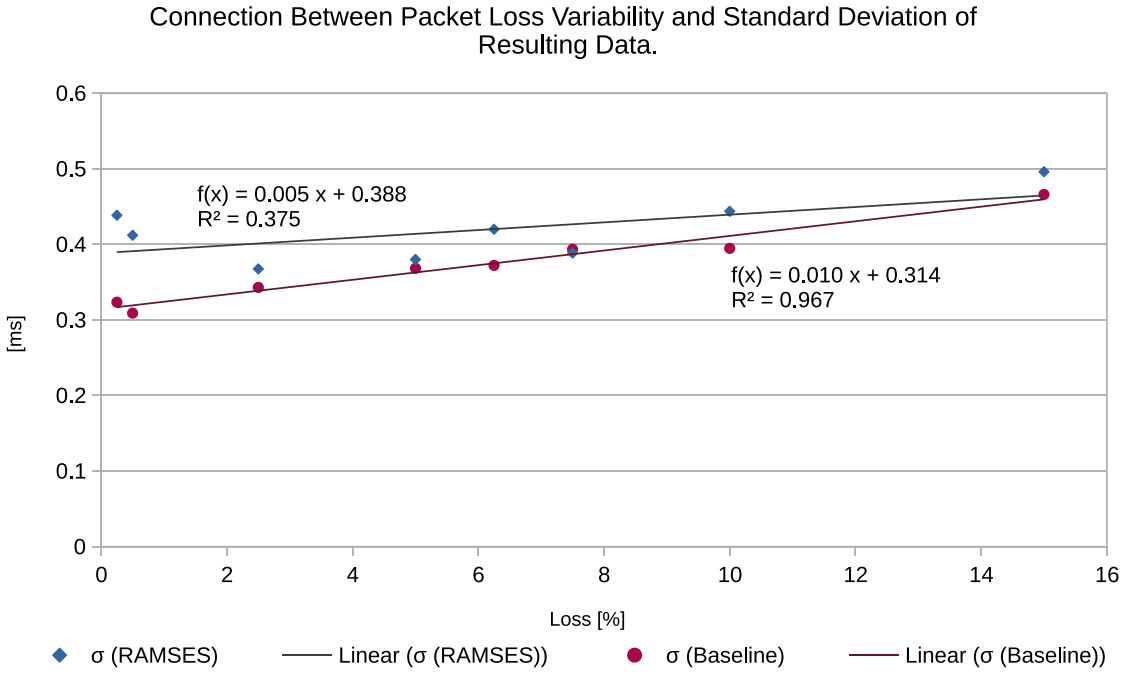


Figure 7.3: Connection between packet loss and standard deviation of the relative time differences between devices with Baseline and RAMSES.

Figure 7.4 contains the mean values of the relative time differences between devices for the two approaches at different packet loss percentages with fitted linear functions. Both Baseline and RAMSES are very close to zero and the R^2 values for the fitted linear functions are very low, indicating that the linear model is not suitable to explain the variations. Our interpretation again is that the mean for both approaches is actually quite constant and the variation seen at different packet loss percentages is due to the limited amount of data used for the analysis.

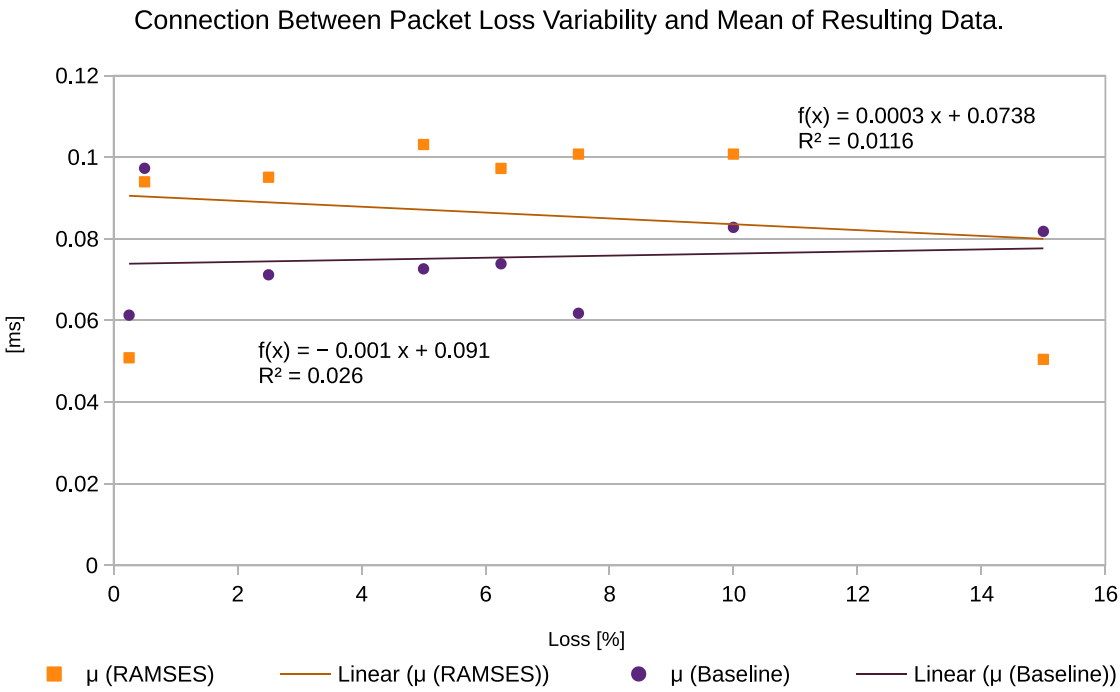


Figure 7.4: Connection between packet loss and mean relative time differences between devices with Baseline and RAMSES.

For packet loss, all outliers were excluded when determining the mean and standard deviation of the results because extreme outliers (>50ms) were found in the Baseline data that would otherwise have distorted the results. When evaluating the RAMSES results, no extreme outliers were found. For the non-extreme outliers, the 1.5*IQR rule was applied. Figure 7.5 shows the number of outliers for both approaches for different percentages of packet loss with fitted quadratic functions. Notably, while with the Baseline approach, the number of outliers increases with packet loss, the opposite is the case when using RAMSES. With our approach, the number of outliers decreases with increasing packet loss. Also, the highest number of outliers for RAMSES is less than half of the highest number of outliers for the Baseline approach.

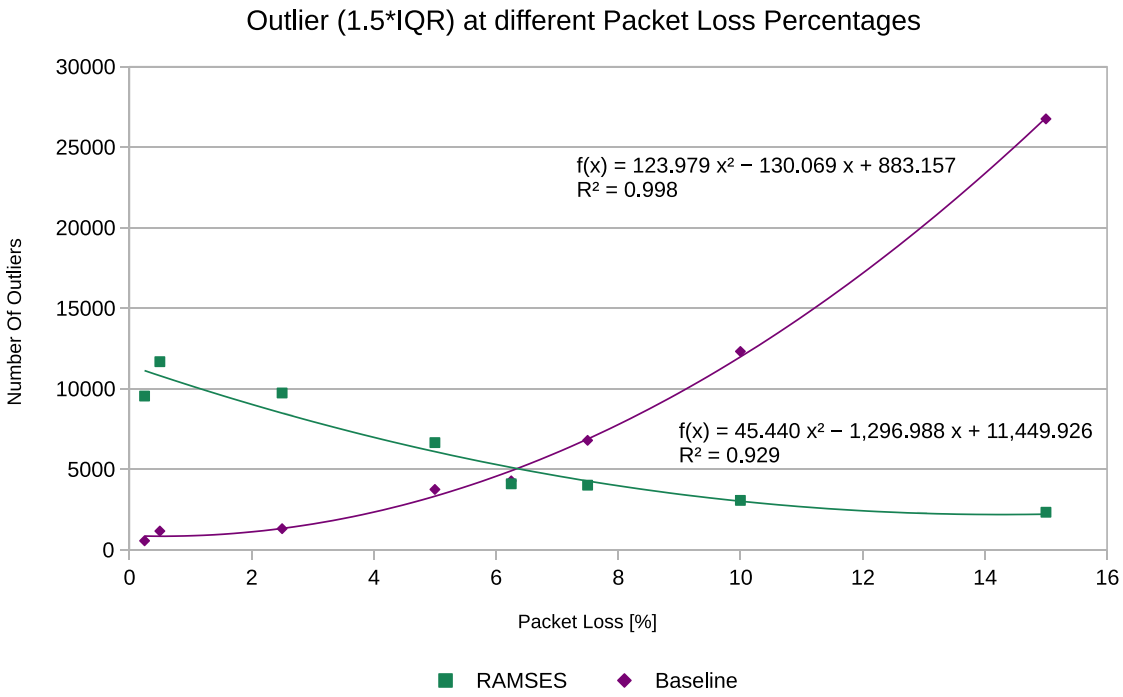


Figure 7.5: Connection between packet loss and number of IQR*1.5 outlier with Baseline and RAMSES.

7.3 Combination of Delay Jitter and Packet Loss

In section 6.3 the results of two asymmetric packet delay and loss combinations of 4ms jitter and 5% loss (medium) and 7.5ms jitter and 15% loss (high) were reported. The F-Statistic values of 15.8 (medium) and 22.09 (high) show a highly significant difference ($p < 0.001$) between Baseline and RAMSES, so RAMSES performs significantly better than the Baseline approach. With our approach, the standard deviation of relative time differences between devices is reduced from 10.81ms to 2.3ms (high), respectively 5.13ms to 1.28ms (medium).

7.4 Limitations

Looking at the performance of RAMSES with packet loss (section 7.2), an improvement over the Baseline approach was made by not having a linear relationship between the amount of packet loss and the standard deviation of the results. However, the standard deviation of the test bench results when using RAMSES is generally slightly higher than the standard deviation of the results when using the Baseline approach. Further analyses and investigations are required to possibly reduce or eliminate the additional variability.

This work focuses on synchronization performance and does not take the consumption of resources like computational power, network communication, and energy into account.

However, the synchronization performance improvement of RAMSES comes at the cost of efficiency. The collection of five on-wire results for the synchronization requires five times more computing power and network communication than the single on-wire result used for synchronization in the Baseline approach. Consequently, it is estimated that RAMSES consumes at least five times more energy than the Baseline approach. Additional computing power is required for averaging and filtering the on-wire results. In case on-wire results are considered outliers and get discarded, extra on-wire protocol executions are performed at the expense of computing power, network communication, and energy consumption. Analyses like the ones carried out in this work for synchronization performance should also be performed for the energy consumption of the approaches.

CHAPTER 8

Conclusion

When multiple wearable sensor nodes are used to collect data, combining the sensor readings is a crucial task. Timestamp-based data combination is a flexible solution for this task but requires accurate time synchronization. To retain the flexibility, a portable time synchronization method that achieves the required accuracy is needed. In this work, portable, application-level software-based time synchronization approaches are researched by analyzing the time synchronization performance of the default time synchronization method of the commonly used microcontroller ESP32 (SNTP), as well as the time synchronization performance of our own approach RAMSES - Repeated Averaging of Multiple SNTP Executions for Synchronization - which uses outlier filtering and statistical analysis to enhance synchronization results. The performance analysis was conducted using our custom-built test bench, with which different amounts of asymmetric packet delay and packet loss were simulated. Long-term measurements of 96 hours were conducted to build statistically robust models to assess the synchronization performance. Short-term measurements of 1 hour were used to validate the models. Long-term and short-term measurements were carried out for asymmetric packet delay and packet loss individually. Results show that our approach performs better with asymmetric packet delay than with packet loss. This should be addressed in future work. Additionally, 3-hour-long measurements with a combination of asymmetric packet delay and packet loss were performed and it was shown that in comparison to SNTP, RAMSES significantly ($p < 0.001$) reduces the amount of time differences between synchronized devices.
RAMSES trades more network communication and thus also more power consumption for better synchronization results. A similar performance analysis as the one conducted in this thesis for packet loss and packet delay is desirable for power consumption. This too should be part of future work. Regardless, based on the results of this work, it is concluded that RAMSES is a suitable and easy-to-use alternative to SNTP, especially in settings where more accuracy is needed.

Übersicht verwendeter Hilfsmittel

Im Schreibprozess habe ich `deepl.com`, `grammarly.com` und `languagetool.com` verwendet, um aus meiner Muttersprache ins Englische zu übersetzen bzw. geschriebenes auf Rechtschreibung und Grammatik zu prüfen.

Für die Literaturrecherche kamen neben `scholar.google.com` auch `semanticscholar.org` und `scienceos.ai` zum Einsatz.

ChatGPT wurde im Schreibprozess zum Finden von Synonymen und Alternativformulierungen verwendet, sowie für die Implementierung und im Rahmen der Datenanalyse zum Generieren von Code Snippets (vor allem für die Python-Bibliotheken Pandas, NumPy und Matplotlib).

List of Figures

List of Tables

Bibliography

[45708] Ieee standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pages 1–269, 2008. doi:10.1109/IEEESTD.2008.4579760.

[AAR+20] T. Ahmed, M. Y. Ahmed, M. M. Rahman, E. Nemati, B. Islam, K. Vatanparvar, V. Nathan, D. McCaffrey, J. Kuang, and J. A. Gao. Automated time synchronization of cough events from multimodal sensors in mobile devices. In *Proceedings of the 2020 International Conference on Multimodal Interaction*, ICMI '20, pages 614–619. ACM, Oct. 2020. doi:10.1145/3382507.3418855.

[Ala14] M. M. Alani. *OSI Model*, pages 5–17. Springer International Publishing, 2014. doi:10.1007/978-3-319-05152-9_2.

[BAL09] D. Bannach, O. Amft, and P. Lukowicz. *Automatic Event-Based Synchronization of Multimodal Data Streams from Wearable and Ambient Sensors*, pages 135–148. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-04471-7_11.

[CY21] P. Chen and Z. Yang. Understanding precision time protocol in today's Wi-Fi networks: A measurement study. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 597–610. USENIX Association, July 2021. URL `https://www.usenix.org/conference/atc21/presentation/chen`.

[DKLM05] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester. *Exploratory data analysis: numerical summaries*, pages 231–243. Springer London, 2005. doi:10.1007/1-84628-168-7_16.

[GED+22] A. J. Goodwin, D. Eytan, W. Dixon, S. D. Goodfellow, Z. Doherty, R. W. Greer, A. McEwan, M. Tracy, P. C. Laussen, A. Assadi, and M. Mazwi. Timing errors and temporal uncertainty in clinical databases—a narrative review. *Frontiers in Digital Health*, 4, Aug. 2022. doi:10.3389/fdgth.2022.932599.

[GLD+24] T. J. Gilbert, Z. Lin, S. Day, A. F. d. C. Hamilton, and J. A. Ward. A magnetometer-based method for in-situ syncing of wearable iner-

tial measurement units. *Frontiers in Computer Science*, 6, Apr. 2024. doi:10.3389/fcomp.2024.1385392.

[HGBR21] R. Hofmann, D. Grubmair, C. A. Boano, and K. Romer. X-sync: Cross-technology clock synchronization among off-the-shelf wireless iot devices. In *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. IEEE, Oct. 2021. doi:10.1109/lcn52139.2021.9524940.

[HGTW24] Y. Han, T. J. Gilbert, X. Tan, and J. A. Ward. The wand chooses the imu - open source hardware for synchronising wearables using magnetometers. In *Companion of the 2024 on ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '24, pages 939–943. ACM, Oct. 2024. doi:10.1145/3675094.3678485.

[HHN+07] P. Hall, Y. Hao, Y. Nechayev, A. Alomainy, C. Constantinou, C. Parini, M. Kamarudin, T. Salim, D. Hee, R. Dubrovka, A. Owadally, W. Song, A. Serra, P. Nepa, M. Gallo, and M. Bozzetti. Antennas and propagation for on-body communication systems. *IEEE Antennas and Propagation Magazine*, 49(3):41–58, jun 2007. doi:10.1109/map.2007.4293935.

[HIK+22] M. Harada, S. Izumi, R. Kozeni, Y. Yoshikawa, T. Ishii, H. Kawaguchi, S. Uemura, and K. Araki. 20-µs accuracy time-synchronization method using bluetooth low energy for internet-of-things sensors. In *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, jan 2022. doi:10.1109/ccnc49033.2022.9700687.

[JLHW11] A. Jurgelionis, J.-P. Laulajainen, M. Hirvonen, and A. I. Wang. An empirical study of netem network emulation functionalities. In *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6. IEEE, July 2011. doi:10.1109/icccn.2011.6005933.

[KHMP16] A. Khan, N. Hammerla, S. Mellor, and T. Plötz. Optimising sampling rates for accelerometer-based human activity recognition. *Pattern Recognition Letters*, 73:33–40, Apr. 2016. doi:10.1016/j.patrec.2016.01.001.

[KK23] S. Kumar and A. R. Kumar. Hardware time-stamping in network time protocol and ntp to 1pps generation. In *2023 8th International Conference on Communication and Electronics Systems (ICCES)*, pages 639–643. IEEE, June 2023. doi:10.1109/icces57224.2023.10192787.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978. doi:10.1145/359545.359563.

[LKK+17] C. Luo, H. Koski, M. Korhonen, J. Goncalves, T. Anagnostopoulos, S. Konomi, S. Klakegg, and V. Kostakos. Rapid clock synchronisation

for ubiquitous sensing services involving multiple smartphones. In *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, UbiComp '17. ACM, Sept. 2017. doi:10.1145/3123024.3124432.

[Man01] S. Mann. Wearable computing: toward humanistic intelligence. *IEEE Intelligent Systems*, 16(3):10–15, May 2001. doi:10.1109/5254.940020.

[MDBS18] S. K. Mani, R. Durairajan, P. Barford, and J. Sommers. A system for clock synchronization in an internet of things. *CoRR*, abs/1806.02474, 2018, 1806.02474. URL `http://arxiv.org/abs/1806.02474`.

[MDC+24] D. K. Ming, J. Daniels, H. Q. Chanh, S. Karolcik, B. Hernandez, V. Manginas, V. H. Nguyen, Q. H. Nguyen, T. Q. Phan, T. H. T. Luong, H. T. Trieu, A. H. Holmes, V. T. Phan, P. Georgiou, and S. Yacoub. Predicting deterioration in dengue using a low cost wearable for continuous clinical monitoring. *npj Digital Medicine*, 7(1), Nov. 2024. doi:10.1038/s41746-024-01304-4.

[RFC1361] P. D. L. Mills. Simple Network Time Protocol (SNTP). RFC 1361, Aug. 1992. doi:10.17487/RFC1361.

[RFC2330] D. G. T. Almes, J. Mahdavi, M. Mathis, and D. V. Paxson. Framework for IP Performance Metrics. RFC 2330, May 1998. doi:10.17487/RFC2330.

[RFC5905] J. Martin, J. Burbank, W. Kasch, and P. D. L. Mills. Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905, June 2010. doi:10.17487/RFC5905.

[RFC958] D. Mills. Network Time Protocol (NTP). RFC 958, Sept. 1985. doi:10.17487/RFC0958.

[RTH20] C. Raman, S. Tan, and H. Hung. A modular approach for synchronized wireless multimodal multisensor data acquisition in highly dynamic social settings. In *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, pages 3586–3594. ACM, Oct. 2020. doi:10.1145/3394171.3413697.

[RW16] C. C. Rheinlander and N. Wehn. Precise synchronization time stamp generation for bluetooth low energy. In *2016 IEEE SENSORS*, pages 1–3. IEEE, Oct. 2016. doi:10.1109/icsens.2016.7808812.

[Sax14] P. Saxena. Osi reference model–a seven layered architecture of osi model. *International Journal of Research*, 1(10):1145–1156, 2014.

[SGO+23] Y. Sun, D. A. Greaves, G. Orgs, A. F. de C. Hamilton, S. Day, and J. A. Ward. Using wearable sensors to measure interpersonal synchrony in actors and audience members during a live theatre performance. *Proceedings of*

the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 7(1):1–29, Mar. 2023. doi:10.1145/3580781.

[SPHS23] A.-K. Schalkamp, K. J. Peall, N. A. Harrison, and C. Sandor. Wearable movement-tracking data identify parkinson's disease years before clinical diagnosis. *Nature Medicine*, 29(8):2048–2056, July 2023. doi:10.1038/s41591-023-02440-2.

[TAA19] F. Tirado-Andrés and A. Araujo. Performance of clock sources and their influence on time synchronization in wireless sensor networks. *International Journal of Distributed Sensor Networks*, 15(9):155014771987937, Sept. 2019. doi:10.1177/1550147719879372.

[WSL+19] C. Wang, Z. Sarsenbayeva, C. Luo, J. Goncalves, and V. Kostakos. Improving wearable sensor data quality using context markers. In *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*, UbiComp '19, pages 598–601. ACM, Sept. 2019. doi:10.1145/3341162.3349334.

[WvLSR21] F. Wolling, K. van Laerhoven, P. Siirtola, and J. Roning. Pulsync: The heart rate variability as a unique fingerprint for the alignment of sensor data across multiple wearable devices. In *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pages 188–193. IEEE, Mar. 2021. doi:10.1109/percomworkshops51409.2021.9431015.

[XGMW17] T. Xu, A. Guo, J. Ma, and K. I.-K. Wang. Feature-based temporal statistical modeling of data streams from multiple wearable devices. In *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. IEEE, Nov. 2017. doi:10.1109/dasc-picom-datacom-cyberscitec.2017.34.

[YTLH19] Z. Yan, R. Tan, Y. Li, and J. Huang. Wearables clock synchronization using skin electric potentials. *IEEE Transactions on Mobile Computing*, 18(12):2984–2998, dec 2019. doi:10.1109/tmc.2018.2884897.

[ZZZ+22] J. Zhang, C. Zhang, Z. Zhang, Z. Su, M. Zou, Y. Luo, and G. Zhang. Review of wearable computing technology: Core technologies, typical systems and research trends. In *2022 Global Conference on Robotics, Artificial Intelligence and Information Technology (GCRAIT)*. IEEE, July 2022. doi:10.1109/gcrait55928.2022.00055.