MONTAN UNIVERSITÄT LEOBEN

Chair of Automation

Master's Thesis

Networked Control Systems With Communication Delays

Gemith Mattathil, BSc

May 2023

MONTANUNIVERSITÄT LEOBEN
www.unileoben.ac.at

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Ich erkläre, dass ich die Richtlinien des Senats der Montanuniversität Leoben zu "Gute wissenschaftliche Praxis" gelesen, verstanden und befolgt habe.

Weiters erkläre ich, dass die elektronische und gedruckte Version der eingereichten wissenschaftlichen Abschlussarbeit formal und inhaltlich identisch sind.

Datum 22.05.2023

Gemith

Unterschrift Verfasser/in
Gemith Mattathil

Acknowledgements

Foremost I would like to thank Dr. Gerhard Rath. Who supported me through the whole process of completing this thesis. Thank you for your invaluable input and help and especially for the encouragement you gave me when things were not going so well.

I would also like to express my gratitude to Prof. Paul O'Leary for facilitating and enabling me to write my thesis at the Chair of Automation.

Thank you Philip Mannas for looking through my work and correcting spelling and grammatical mistakes. The Inputs about how to structure and reference different parts of the thesis were very helpful.

I would like to thank my parents. Who supported me in any way they could. Thank you for cheering me up and motivating me to give my best efforts to this work.

Thank you to www.RandomNerdTutorials.com for code examples and knowledge of microcontrollers. I have learned a lot from you about programming microcontrollers.

Kurzfassung

Im Internet kann es aus verschiedenen Gründen wie Netzwerküberlastung, Entfernung zwischen kommunizierenden Geräten und Bandbreitenbeschränkungen zu Verbindungsverzögerungen kommen. Dies kann zu vielen Unannehmlichkeiten und Schwierigkeiten für Benutzer, die über das Internet kommunizieren, führen. Für digitale Steuerungssysteme wird es zur Herausforderung, Prozesse in Anlagen über das Internet mit Steuerungen angemessen zu steuern und zu regeln.

Diese Diplomarbeit konzentriert sich darauf, einen geeigneten digitalen Regler mit geschlossenem Regelkreis zu finden, um einen Prozess durchs Internet trotz der normalerweise vorhandenen Verzögerungen zu steuern und zu regeln. Dazu werden zunächst Internetverzögerungen beobachtet und bewertet. Diese werden von verschiedenen Standorten aus gemessen werden.

Die Zeitmessungen der Verzögerungen wurden über verschiedene Entfernungen, einschließlich inter kontinentaler Verbindungen gemessen. Es wurde versucht, in ähnlichen Uhrzeiten über einen MQTT-Broker zu messen. Diese Messungen wurden anhand von Histogrammen ausgewertet. Dadurch ist es möglich, die vorhandenen Verzögerungen zu beobachten und Korrelationen zwischen den Verzögerungen und den Messparametern zu finden.

Ein Experiment mit einem PI-Controller, programmiert in MATLAB®, wurde eingerichtet, um ein System zweiter Ordnung in Form eines Schaltkreises mit Kondensatoren und Widerständen zu steuern. Die Kommunikation zwischen Steuerung und Anlage erfolgt über das Internet, was zu Verzögerungen bei diesem Informationsaustausch führt. Zusätzlich erfolgt beim Informationsaustausch eine Konvertierung der Kommunikationsprotokolle des Internets UDP und TCP. Die Komponenten des Zustandsvektors wurden mit einem Luenberger Beobachter ermittelt und im PI-Controller benützt. Der Fehlerwert wurde aus der Rückkopplung der Anlage berechnet. Dies liefert einen Hinweis darauf, ob das System wie erwartungsgemäß im Betrieb ist.

Abstract

On the internet, it can come to connection delays due to different reasons, like network congestion, distance between communicating devices and bandwidth limitations. This can lead to many inconveniences and difficulties for users communicating through the Internet. For digital control systems, it becomes challenging to manage and regulate processes in plants adequately with controllers that are connected over the internet.

This thesis focuses on finding a fitting closed-loop digital controller to manage and regulate a processing plant on the internet despite the normally present delays, by first observing and evaluating internet delays by measuring these from different locations.

The time measurements of the delays were measured over different distances, including cross-continental connections, and attempted to be during similar time periods through an MQTT broker. These measurements were evaluated using histograms to observe the existing delays and find correlations between the delays and the measurement parameters.

An experiment including a PI controller, programmed in MATLAB® had been assembled to control a second-degree plant in the form of a resistance-capacitor-loop. The communication between the controller and plant is established through the internet, which causes delays in this information exchange. Additionally, there is a conversion of the communication protocols of the internet UDP and TCP during the information exchange. The missing components of the state vector were estimated with a Luenberger Observer and used in the PI controller. The error value was calculated from the return signal of the plant. It provided an indication of whether the system was operating as intended.

Contents

Chapter 1

Introduction

This thesis will start with literature research, which is split into two parts. The first part will look at how communication is established through the internet. The focus of this part will be the definition of speed on the internet, the possibilities for the measurement of said speed and reasons for delays during communication on the internet. This part will specifically look at how communication is established through MQTT and UDP because these methods of communication will be considered to be used in the experimental parts of the thesis.

The second part of the literature research will be focused on controllers to control plants through the internet. The main functionality and different types of controllers are discussed in this part. The possibilities of controllers with a feedback loop to handle delay will be the main discussion because this will provide the basis for the main experiment performed in this thesis.

The experimental part is also split into two parts. The first part will take a look at measurements conducted to determine delays during communication through the internet caused by geographical distances. The measurements will be compared graphically. Special attention will be given to the outlier measurements, which represent the worst-case scenario.

The second part, which is the main experiment of this thesis, looks at producing a closed-loop controller that can sufficiently control a process plant despite delays caused by communication through the internet. These are more precisely known as networked control systems, which refer to a closed-loop control system, whose components, for example, controller, sensor and actuator, communicate through a shared common network. This shared common network is the internet in our case [1].

The goal of this thesis is to design a reliable and robust control system, that operates a process over a network, which is affected by delays. The control system should be able to adapt to the latency that the connecting network is afflicted with.

Chapter 2

General Connection Establishment in the Internet

The data communication on the internet is governed by multiple types of physical mediums, such as wires, radio waves or optical fibers, and low-level mechanisms called protocols. This is done by encoding digital values primarily into electrical signals. These connections have especially been important for embedded systems to communicate through the internet, generally known as the ”Internet of Things” (IoT) [2] [3].
The general structure of the internet, the establishment of connections, and the possible causes of delays in the internet will be discussed in this chapter.

2.1 Communication

Communication on the internet is established between a source and a destination. The physical process, terminologies required software and hardware, and how the two devices are connected through the internet will be looked at in this section.

2.1.1 Overall Structure of the Internet

Information is converted into binary format in computers. This format is pragmatic for information transportation because it is easier to detect if there is a signal, one in binary, or if there is no signal, zero in binary. It is also the basic language of CPUs and microcontrollers. This format is also more robust against disturbances, compared to an encoding using more than two states.

The information can be transported in two ways. The first is called baseband transmission. Here the signal uses the whole available frequency spectrum of the transfer channel. An example is Ethernet, the traditional wired connection of devices, in a Local Area Network (LAN). The second method is carrier band transmission, where the original spectrum of the signal is shifted by modulating a carrier. The carrier wave is only slightly

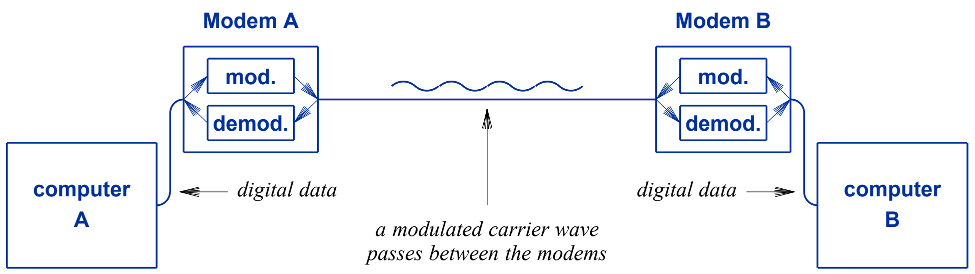

Figure 2.1: Communication between two computers modulated and demodulated by two modems

[3]

modulated. These can be electromagnetic, radio waves or other oscillation forms of energy. The reason for using an oscillating form of energy is that these travel farther than a simple on-off signal. As an example: The sender modulates the carrier wave before sending it to the receiver and the receiver detects the difference compared to the unmodulated carrier wave and demodulates the information from the carrier wave.

The modulation and demodulation are performed by the modem. It is a device that is connected to the internet between the sender and receiver (see Figure 2.1). The internet can be seen as a connection of devices. The sending and receiving devices are identified as hosts. These can be identified as end devices, which is the reason they are also known as endpoint devices, where the information is sent from or the final destination the information is sent to. All intermediate devices are identified as routers. They only have the task to forward the information that is sent to them. The step in the transmission of data from one network device to another is known as a hop [3].

Information is sent through the internet by dividing them into multiple packets. A packet contains the information or parts of it, with metadata attached to it. This metadata includes source-, destination-address and other information and is contained in the packet header [2].

The Internet connects multiple hosts. To enable all hosts to communicate seemingly as simultaneously as possible the process of packet-switching is used. It ensures that packets sent by different hosts take turns to travel to the next router or host so that communication can commence and occur seemingly parallel without having to wait for another to completely finish. There are different types of packet-switching technologies. Depending on the speed, distance to the next router and economic cost, they can differ in capable packet size and how the recipient is identified.

Depending on the distances between communicating hosts, we differentiate between local area networks (LAN) and wide area networks (WAN). LAN is a collection of different devices in close proximity, for example, a home network or an office building. The intermediate devices for a LAN network are switches and routers, these devices will be further discussed in Chapter 3 for their different accesses of the OSI Model Layer. The two most important technologies that are used in LAN connections are Ethernet and Wi-

Fi (Wireless Fidelity). An Ethernet wired connection and Wi-Fi are used for wireless connection to the internet. A Wi-Fi connection between a device and the internet is established by connecting the intermediate device wireless access point, also known as a wireless router, through an Ethernet connection to the modem, which is connected to the internet. It is possible to connect multiple devices to the internet through one wireless access point. The devices and the access point are connected by radio waves. In Figure 2.2 multiple different devices are connected through a router followed by a modem to the Internet Service Provider (ISP).

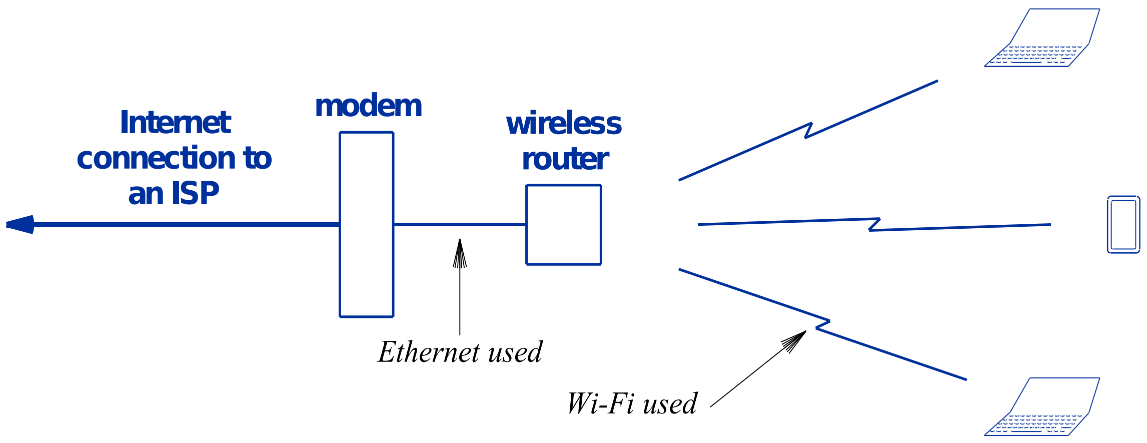

Figure 2.2: Connection devices to the internet through Wi-Fi [3]

WAN covers a geographically wider area of networks. It connects multiple LANs with each other using WAN switches. These LANs can be seen as sites with long distances between them (see Figure 2.3). For devices, be they end devices or intermediate devices, to be identified, each of them has a unique address. This address is called an Internet Protocol address, short IP address. There are mainly two types of IP addresses used on the internet, IP version 4 (IPv4) and the newer IP version 6 (IPv6). IPv4 consists of 32 bits. It is written in four decimal bits. The possible amount, of a little more than four billion addresses, is not sufficient for the current needs of the internet. So the newer IPv6 was introduced, which is a 128-bit address made with the hexadecimal number system. IPv6 can produce $3.4 \cdot 10^{38}$ addresses [3] [4].

With the help of IP addresses, packets can be sent to the correct device through the internet. The IP addresses of the sender and receiver are included in the packets as metadata. IP addresses are usually temporary and are given to the devices from the ISP through the Dynamic Host Configuration Protocol (DHCP). A fixed address for a device that never changes is its unique media access control address (MAC address). It is assigned to the device by its manufacturer and consists of six groups of two hexadecimal digits. This information is added to the metadata if packets are sent in WAN [3] [5].

Just 132 bits addresses are used to identify devices on the internet 16-bit protocol port numbers are used to identify which applications or services are communicating between

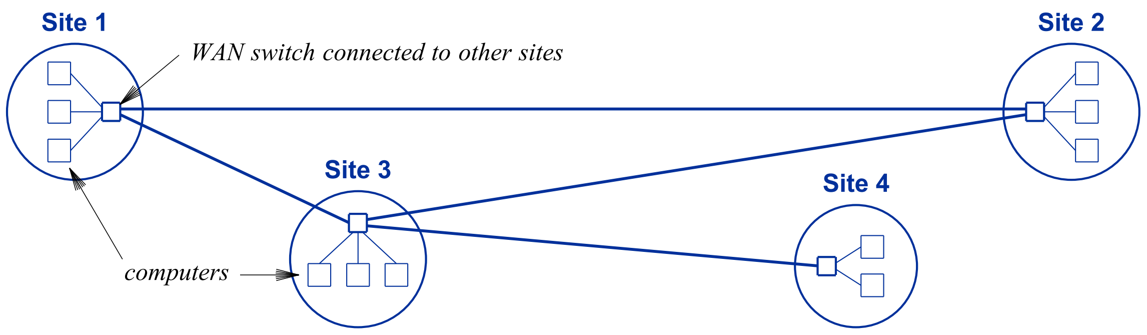

Figure 2.3: Connecting multiple sites through WAN
[3]

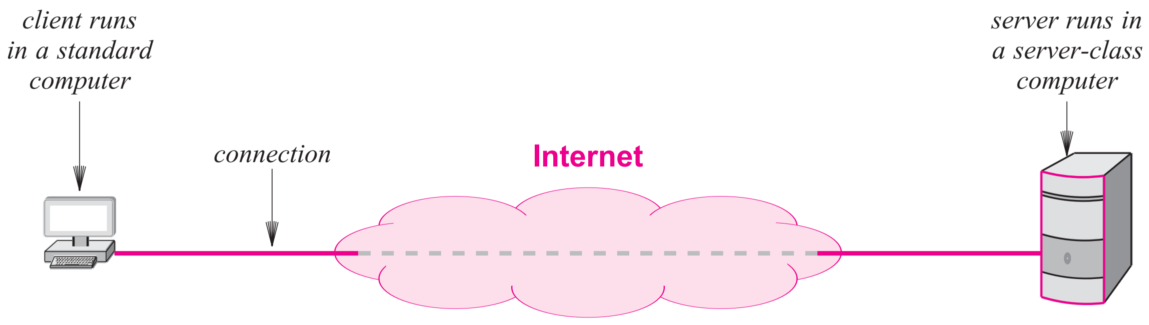

Figure 2.4: Communication between a client computer and server
[2]

client and server. As an example, port number 465 is used for emails via SMTP (Simple Mail Transfer Protocol).

For two devices to communicate through the internet, certain rules have to be followed, these rules also apply to the intermediate devices. For example what format should the message have, what voltage should be used during the connection and how should contact be established between the source and destination? These rules provide the basis for interoperability between devices. The terminology for these rules is protocols. Protocols specify low and high-level details (see Chapter 3) and also responses to errors. Examples of protocols are Hypertext Transfer Protocol (HTTP) and file transfer protocol (FTP) for usage with internet browsers [2] [3].

Communication on the internet is established on the client-server model. Correspondence between devices occurs more specifically between applications of these devices. For example, the web browser on the computer communicates with the application of the website server, to call upon the website hosted by the website server. Hereby the client is the web browser of the computer that demands information and the website server is the server that offers the requested information (see Figure 2.4). The servers must always run to provide services, client devices are only active while requesting information [2] [3].

2.1.2 Connection Establishment and Transport Protocols

A connection on the internet is usually based on the client-server model. The client is the device that initiates the connection request. The server is required to be active to establish the connection, it awaits a request for communication. The server that the client is trying to connect with is identified by the IP address the client requests. After establishing the connection, two-way communication is possible and the required tasks of the client are performed. The connection may be terminated after the tasks are completed. Port numbers are used to identify which service is required from the client [3].

The sending of information between these two devices is handled by transport protocols. These protocols split up the information packets and determine the attached metadata before sending them to their destination. Examples of transport protocols are User Datagram Protocols (UDP) and Transmission Control Protocols (TCP). These high-level protocols are combined with the IP, which adds the source and destination addresses to the metadata of the packets, so the intermediate devices know what the destination is [5].

UDP is a fast, but not reliable transport protocol. It adds a port number to the packets. This makes it clear from which software or service the information was produced and to which software or service the information will be used at the destination. UDP does not monitor if the packets arrived at the destination, because of this the source will not resend any lost packages. The order of the packages can also arrive disarranged at the destination. Examples of services that use UDP are video telephony, video streaming and online gaming. These services prioritize speed instead of reliability [3].

TCP is a reliable protocol that checks if the sent packages have arrived at the destination, if packages are lost in transmission it will resend these. The downside to this protocol is that transmission speed is low. TCP creates a communication channel between the source and destination called a pipe or stream to send data. The sent packets are numbered by TCP in the correct order and sent individually. The destination has to send back an acknowledgment message after receiving a packet. If there is no acknowledgment message from the destination, the next packets in the series have to wait for the lost packet to be resent and acknowledged [3].

To start the transmission TCP has to execute a three-way Handshake between the source and destination. To build the information stream TCP sends special packets between the source and destination. The packets are known as SYN-SYN/ACK-ACK, SYN for synchronization and ACK for acknowledgment. First, the source sends a SYN packet to the destination. The destination returns a SYN packet of its own, with an ACK message. At last, before the actual packet is sent the source sends an ACK message to the destination (See Figure 2.5). This three-way Handshake is executed before and after each sent packet. The transmission is also ended with a special packet FIN-FIN/ACK-ACK, FIN for finalized. For more information on TCP/IP see Chapter 3.

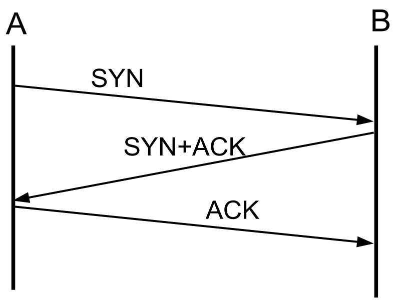

Figure 2.5: Three-way handshake of TCP
[6]

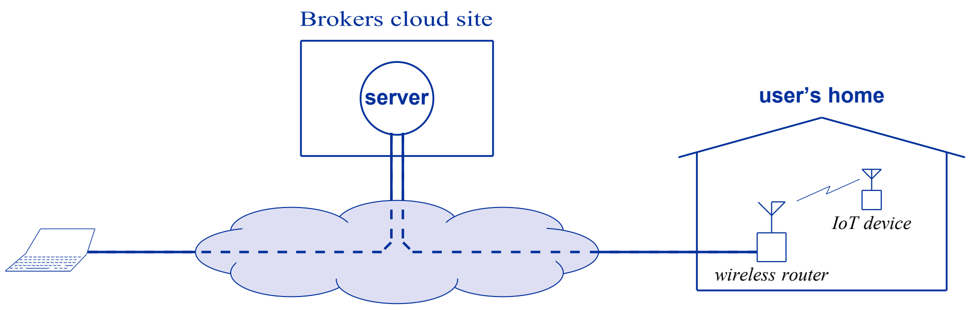

Figure 2.6: User communicating with IoT Device through broker
[3]

If an IP address is temporary it cannot easily be accessed from the outside of its LAN, which comprises of all devices connected to the router. This is why servers have permanent IP addresses. For Internet of Things (IoT) devices, that have temporary IP addresses, this is problematic. To solve this cloud application services are used. A server is used as a broker between the IoT and the control application. This server has a permanent IP address. The IoT device contacts the server and the user can access controls or information from the IoT device through the server (see Figure 2.6). Both the user's control device and the IoT device have to provide the same user ID to the server to communicate with each other through the server [3].

2.2 Internet Speed

The expression speed can be misleading. All data travel at about the same speed. There is only a difference in speed depending on the physical medium the data travels through. For example, data travels faster through optical fiber cables than normal cables made out of copper. When talking about speed on the internet, usually the network capacity, also known as throughput, is meant by this. Capacity is measured by bits per second. On the internet, the dimensions of data transported per second are in the millions and billions. So the typically used units for capacity are Megabits per second (Mbps) and Gigabits per second (Gbps). Fast communication is vital in data exchange of real-time systems. For example, voice over landlines, mobile phones, video and teleconferences [3] [7].

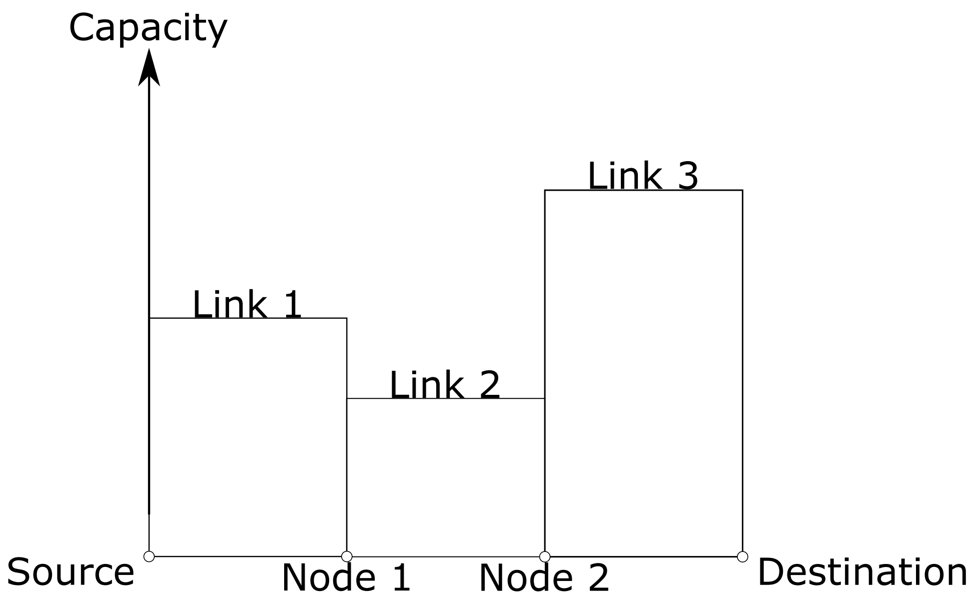

Figure 2.7: Capacities between source and destination and the limiting link

Even so, the physical distance between two communicating devices has a substantial influence on the speed of their connection. The further away the destination of the data is, the longer it will take to reach this destination.

Capacity can be further specified to end-to-end capacity. As discussed in section 2.1.1 the information usually travels through multiple nodes before reaching its destination. The end-to-end capacity is defined as the minimum capacity between two nodes on the route between the source and destination. Even if a link between two nodes has high capacity, it does not mean that the whole capacity can be used by the user. As an example in Figure 2.7 Link one and link three have high capacities, but the limiting link is the second. The capacity of the second link is also the end-to-end capacity between the source and destination. The link is also used by other users to communicate, so the capacity between the nodes has to be shared. The capacity that is free is called the available bandwidth. The end-to-end available bandwidth is analogous to the end-to-end capacity the minimum available bandwidth on the route between the communicating devices. The Network can become congested if too many users are utilizing the same link between nodes. This leads to very low capacity and the traffic between these nodes ”slows down”. So it is not possible for data to be transported faster than its slowest network participant, with the least capacity. This slowest participant is called a bottleneck [3] [8] [9].

Capacity is also dependent on the path taken between two devices. The Border Gateway Protocol (BGP) is responsible for the chosen path. Depending on the path taken, the cost can also vary. The ISP that is responsible for the devices chooses the path with the highest capacity and also the cheapest one. The calculation of the costs depends on the number of packets transferred, this is known as the transit fee. So the cost of communication is also a factor in how fast data is transferred [5].

A usual bottleneck for communication is located in the so-called ”last mile”. This is the connection between a device and its ISP’s network. It can be for example the device connecting to the router, be it wired or through Wi-Fi, or for mobile devices connecting to the nearest cellular tower. Wired connections are faster than connections through radio waves [3] [10].

2.3 Reasons for Delay

Delay is principally measured in units of time, usually in milliseconds, it takes for one packet to travel from its source to its destination device. This amount of time is defined as latency. Latency can have its cause in hardware or software. These causes will be discussed further in the following subsections. An in-depth look at delays in Networked Control Systems (NCS) will be analysed in Chapter 6.

2.3.1 Hardware

As mentioned above one of the main reasons for delay is the physical distance between two communicating devices. The delay is defined by the medium used to transport the information and the distance traveled by the data. This distance is usually not a straight line between two endpoint devices, as can be seen in Figure 2.8 which shows the route data travels for communication between Munich and Bratislava. It can be seen that the network distance is more circuitous than the geographical distance. Longer distance also means more intermediate devices that are used to communicate, this usually in turn means more hops, which causes the time for information to travel to and from the endpoint devices to be longer. Sometimes adding hops can be beneficial to avoid older intermediate devices that would slow down the speed more than more hops over newer devices as discussed below. The total number of hops in a connection is known as the hop count. In conclusion the longer the distance between Endpoint devices the longer the delay [3] [11].

The Hardware responsible for connecting a device to the internet is commonly known as a Network Interface Card also known as Network Interface Controller (NIC). Depending on the model of the NIC the processing speed can vary. This can be a reason for delay in the sending and the receiving end of a communication [3].

Another reason for delays can be the middleboxes, also known as middleware. These are devices between two communication points, that are not responsible for packet forwarding. Tasks of middleboxes include security, such as firewall and malware detection, load balancing and translation functions. These tasks can lead to a bottleneck during communication [10] [12].

The version of devices in a network can become a bottleneck to the internet speed. The devices communicating over the internet are always backward compatible in terms

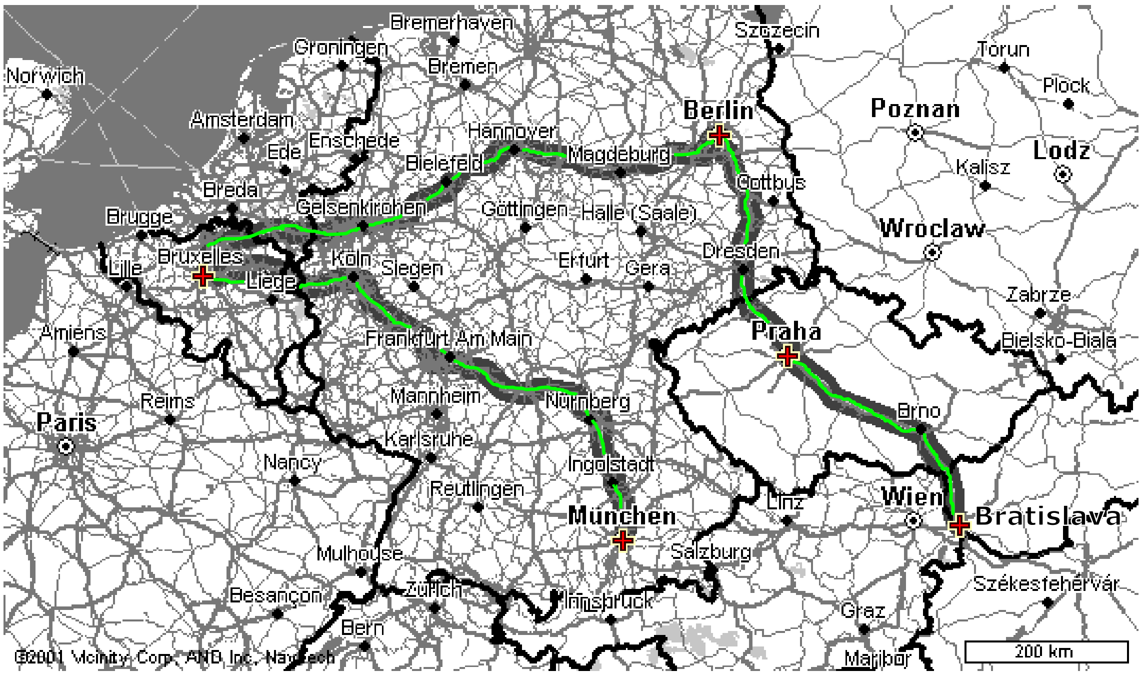

Figure 2.8: The route that data travels between Munich and Bratislava [11]

of connection speed. Consequentially, to avoid packet loss (see Section 2.3.2), the possible highest speed is defined uniformly over the entire connection. More often than not the older a device the slower its internet communication capabilities are. So if an older endpoint device is connected to a network running with the latest devices, the devices that are part of the Network will negotiate a common network speed that is acceptable for all partaking devices. In this case, the older endpoint device will be the limiting factor for the speed of the connection. This also applies to the versions of the middle devices [3].

Errors can cause packets to be resent, which in turn slows down the internet speed. Errors can be caused by packets that are damaged by electrical and/or magnetic interference during transportation. These types of disturbances can be minimised by using shorter connection distances. A shorter distance picks up less interference and also reduces the chance of interference occurring. Additionally as mentioned above the speed increases because the traveled distance is shorter. Another possibility to reduce electrical and/or magnetic interference is by using coaxial cables. These cables enclose the wire through which the data travels with a metal shielding [3].

2.3.2 Software

Delays caused by the hardware of the router are negligible, if the router is working as intended. But the router can be congested, this is a not easily predictable reason for delays that can cause temporal fluctuations in the speed of the internet. These occur when data that flows into a router, or other devices in the network chain, can not be forwarded fast enough and the buffer of the device overflows. This can also lead to packet loss or even complete connection failure. Packet loss can be described by the packet loss rate, which is the number of packets that are lost divided by the total number of packets that were sent during a connection session. TCP detects packet loss when the acknowledgement packet of the receiver does not return to the sender's device within the retransmission timeout (RTO) window. RTO is dynamically determined during the TCP connection. After RTO TCP sends the packet with the missing acknowledgement message again [3] [10] [13].

A way to avoid packet loss and slowing down of the data transfer due to congestion is TCP. If a node in the communication chain receives more data than it sends out, its buffer is being filled with data it can not forward. When this happens the node is congested. TCP can reduce the rate at which data is sent to this node and thus avoiding overflow and giving the node time to send the data in its buffer. After the buffer is cleared TCP increases the rate of data sent to the node to its default value [3]. For more information on TCP see Chapter 3 and Section 2.1.2.

Another effect of internet delay is jitter. This is caused when packets arrive at their set destination with too much time delay or out of order during a stream of data. The travel time of the delayed packets varies from the mean travel time of the other packets. This can typically be observed in video or audio calls, where the video or audio becomes incomprehensible or stutters [3].

The receiving window also has an effect on the internet speed. The receiving window is the size of the buffer that a device in the connection chain has to store data. The difference depending on the receiving window (Rwnd) can be seen in Figure 2.9. Every dot shows an individual measurement of Internet speed and RTT (Round-Trip Time, see also Section 2.4) and receiving window size of one user. The buffer sizes of the receiving windows represented are 16, 64, 256 and 512 KB. A typical Windows XP box used to have a window size of 64 KB. The y-axis shows the internet speed and the x-axis shows represents the RTT, which is the time needed to send a packet to the endpoint and receive it back at the sender's device [8].

To provide a reliable and fast internet speed, it is not sufficient enough that the hardware of the devices used in a network are the latest, but the software of these devices have to also be up to date. For example, the drivers and middleware used in routers should be the latest releases from the manufacturers to improve internet speed [3] [12].

2.4 Measurement Methods

Important metrics for measurements for end-to-end internet performance, meaning the performance of the internet connection between sender and receiver, are throughput (see Section 2.2), latency, also known as delay (see Section 2.3), and packet loss (see Section 2.3.2) [10].

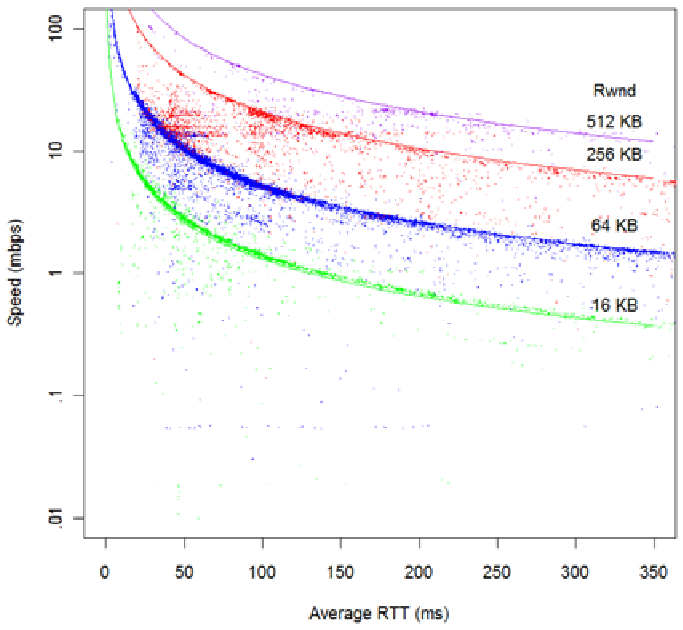

Figure 2.9: Difference of internet speed depending on the size of the receiving window [8]

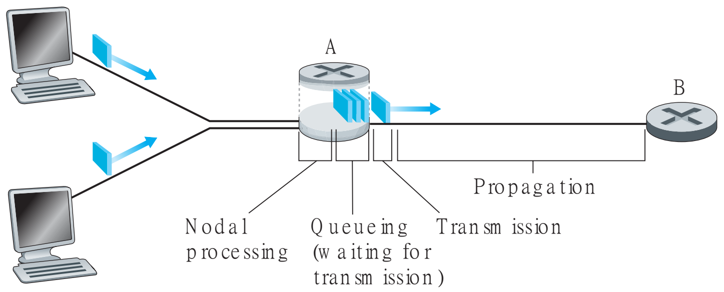

Figure 2.10: The four types of delays displayed for a router [14]

The resulting delay that exists during communication between two devices can be split up into four main components. The delay that is being looked at here is the end-to-end delay. This delay is not commonly referred to when discussing delays on the internet. The most used metric for delays is the round-trip time (RTT), the reason for this preference will be discussed later in this section. The four main components of the end-to-end delay are processing delay, transmission delay, propagation delay and queuing delay. See Figure 2.10 for the four types of delays represented from a viewpoint of a router [11].

Processing delay is the delay that comes from processing the data at the device which is part of the communication chain. It depends on the hardware of the device and the tasks required to forward the packet to the device. It can further be split up into two parts, a deterministic part, which stays the same for a device, and a stochastic random part, which can vary depending on the tasks required to be performed by the device for forwarding

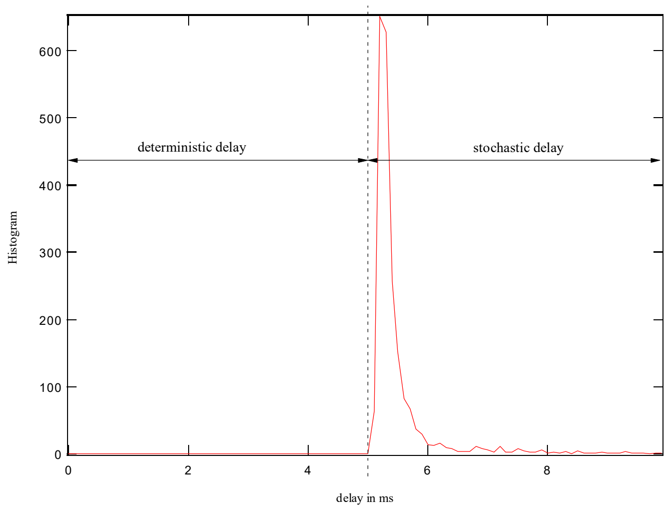

Figure 2.11: The deterministic and stochastic parts of a histogram of delays. Link from Amsterdam to London

[11]

the data [11].

Transmission delay is measured by the sum of the time it takes for a packet to travel from one device's sending port to the other successive device's, on the communication chain, receiving port. It is defined by the link speed, which is the physical speed data travels through a wire or through radio waves. It is important to notice that this delay is the time to transport the complete packet, so from the first bit to its last and not just one bit of data [11].

Propagation delay is the delay it takes to transport one bit through the physical medium. For example, it would take longer for a bit to travel from an earth-bound device to a satellite, than to travel through an optical fiber cable [11].

Queuing delay is dependent on the state of the buffer (see Section 2.3.2) on an intermediate device. This is a purely stochastic type of delay. It depends on the hardware of the intermediate devices and more so on the state of the internet traffic. If there are high amounts of traffic chances are higher that there will be more congestion and with this more filled buffers for the intermediate devices, which in turn leads to high queuing delays [11].

As mentioned for processing delays, there are deterministic and stochastic delays. Transmission and propagation delays are seen as deterministic delays. Queuing delays are stochastic in nature. By graphing a histogram of tests over delays one can see the deterministic and stochastic parts of the measurements. Figure 2.11, which represents measurements between Amsterdam and London, shows this behavior. The Graph in Figure 2.11 also shows a typical form of delay measurements represented in a histogram, which is a Gamma-like shape with a heavy tail. This shape can also be observed in most of the measurements done in the experimental part of this thesis (see section 7.2) [11].

The round-trip time (RTT) is the time it takes for data to travel from the sending device to the receiving device and back to the primary sending device. This method is preferred to the measurement of end-to-end delay because it is more unambiguously due to the omission of the need for time synchronization of both sending and receiving devices. With RTT the delay is calculated using the same clock of the sending device. The widely used Ping command is utilized to measure the RTT of one device to another [10] [13].

Usual speed measurement tools use the active measurement test to produce results. The tests introduce new traffic, called probe traffic, into the network and measure the characteristic metric from this artificially fabricated traffic. Another approach to measuring these properties is the passive test. This observes the already existing network traffic without introducing new traffic of its own. Active measurements provide a more accurate result than passive measurements, but it also disrupts the network traffic of the system by introducing traffic of its own. Additionally, it may not be possible to intentionally introduce new traffic into some systems, because of access and security restrictions [10].

Typically TCP is used for speed tests. The reasoning behind this is that most internet applications use TCP to communicate over the internet and so it is a good representation of the user experience. TCP is also a reliable connection-oriented protocol, this makes sure that the packet is received by the receiving device and if it is not received TCP retransmits the package (see Chapter 3 for more on TCP) [10].

The duration of a speed test is important for receiving adequate results that reflect the health of the network connection as close as possible. Measurements should last for a long period of time. Measurements conducted by the average user of their internet speed are prone to be affected by selection bias. The reason for this is that users usually measure their speed when there seems to be issues with their connection, so typically when the internet speed is slow. This is not a good representation of their average internet speed.

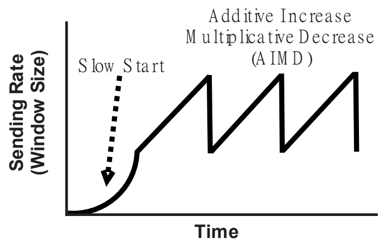

Figure 2.12: Illustrative Graph showing the slow start and AIMD phase of TCP [10]

Another reason for speed tests to be of a longer duration is associated with TCP itself. After the three-way handshake (see Section 2.1.2) TCP enters a phase called slow start. In slow start, the protocol probes the connection to determine adequate transfer speed. It does this by slowly ramping up the amount of package sent until either it comes to packet loss or the sender window size is reached (see also Section 2.3.2). This is a congestion control mechanism of TCP. This part of the data transfer is not representative of the actual transfer speed in its steady state phase, which is called Additive Increase Multiplicative Decrease (AIMD) and is the far majority in a long communication session through the internet. The slow start and AIMD are shown in Figure 2.12, it can be seen that AIMD constantly varies, this is TCP trying to find spare capacity during the data transfer [10] [13].

To shorten the duration and to have a more representative measurement of current web browsers multiple TCP connections are opened during a speed test. From [10] there should at least be four TCP connections for accurate measurements of throughput. Using multiple TCP connections can shorten the duration of tests from 100 to 10 seconds [10] [15].

Chapter 3

OSI and TCP/IP Model

The main models used to describe communication through the Internet are the Open Systems Interconnection (OSI) model and Transmission Control Protocol/ Internet Protocol (TCP/IP) Model. The basis for these models was created in the late 1960s called the Advanced Research Projects Agency Network (ARPANET), or more precisely ARPANET Reference Model (ARM), whose main task is coordinating packet switching in WANs (see also Section 2.1.1). These two models are called protocol suites or families, which are split up into layers, that are activated in a linear fashion and are responsible for different facets of the connection. These layers contain protocols, which have different tasks during internet communication. The layers are ordered from seven to one for the OSI model and five to one for the TCP/IP model. Each layer serves the layer above it and gives tasks to the layer below. These protocol suites and the TCP and UDP protocols will be discussed in this chapter [2] [5] [13].

3.1 OSI Model

The International Organisation for Standardization (ISO) published the OSI model in 1984. It consists of seven layers, which can be seen in Figure 3.1. These layers can be seen as groups. Layer seven to layer four are executed on the host device and layers three to one are executed on network devices. The OSI model is rarely used, it serves as a reference and a framework for educational purposes, because it splits the layer more clearly than the TCP/IP protocol suite [2] [13].

Layer seven is the application layer. This is the layer the user interacts with directly. An example of an application on a device is a Web browser. Typical well-known protocols used in the application layer are Hypertext Transfer Protocol Secure (https), Simple Mail Transfer Protocol (SMTP) and File Transfer Protocol (FTP) [5] [13].

Layer six is the presentation layer. The task of this layer is standard encoding/decoding of information and format conversion, for example, compression of audio and/or

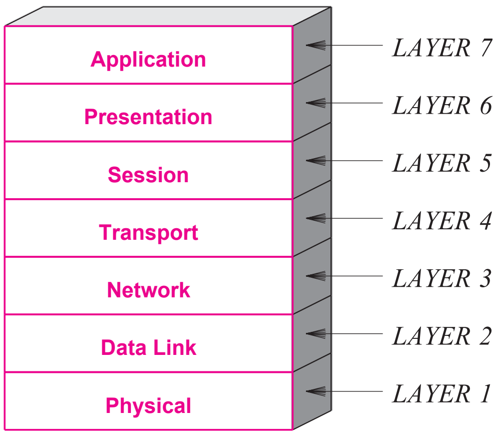

Figure 3.1: The 7-Layer of the OSI Model
[2]

video. Some protocols that are used in this layer are American Standard Code for Information Interchange (ASCII), Moving Picture Experts Group (MPEG) and Portable Network Graphics (PNG) [5] [13].

Layer five is the session layer. This layer is tasked with managing sessions. These sessions are interactions between applications on the same device. Some protocols used in this layer are Remote Procedure Call (RPC), Session Initiation Protocol (SIP) and Network File System (NFS) [5] [13].

Layer four is the transport layer. Protocols of this layer are responsible for the transmission of data. Examples of protocols are UDP and TCP, which will be discussed further in Section 2.1.2 [5] [13].

Layer three is the Network layer. This layer is responsible for the addressing schemes, for example, IP addresses, which dictate the routing of packets over the internet. Protocols of this layer are Internet Protocol (IP), Address Resolution Protocol (ARP) and Border Gateway Protocol (BGP) [5] [13].

Layer two is the data link layer. This layer is tasked with managing the transport of packets between two directly neighboring devices. It uses MAC addresses to communicate with nearby devices in the same LAN which use the same medium, for example, Ethernet. Some protocols of this layer are Ethernet, Wi-Fi and Bluetooth [5] [13].

Layer one is the physical layer. This is the layer that defines the specification for the physical data communication, for example through phone lines and fiber-optic cables. It sends and receives raw bit streams. It can also detect and correct low-level errors. Some protocols of this layer are Fiber Optic, Universal Serial Bus (USB) and Thunderbolt [5] [13].

The devices between a communication connection through the internet, also known as intermediate devices, access different layers when they forward a packet. The protocols of the layers add information to the header of the packets. This information assists the intermediate in forwarding the packets [5] [13].

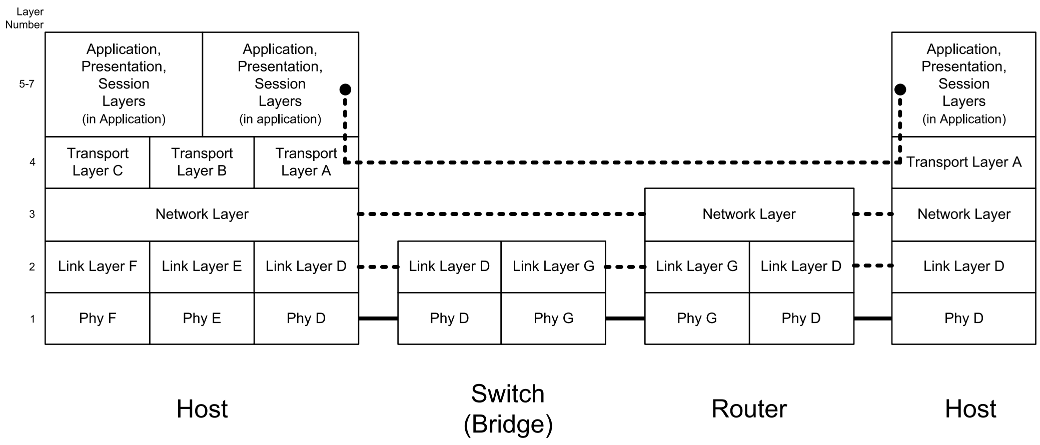

Figure 3.2: Layers of the OSI model used by devices
[13]

The simplest of these devices are network adapters, repeaters and modems. These devices work on layer one, the physical layer, which provides the device with the necessary information about their neighbouring devices to forward the packet [5].

Another device that transports packets is a switch. It is often used in LAN connections. Packets are exactly sent to the receiving end device through a switch, this is why it is considered an intelligent device compared to a hub which would broadcast the packet sent to it to all connected devices. The switch does this by accessing the MAC address in layer two, the link layer [13].

A more sophisticated connecting intermediate device is the router. This device is used to connect to the internet from a LAN connection. The layer accessed by the router is layer three, the Network layer. It uses the IP address to forward packets to their destination [13].

Nowadays these rules for switches and routers can differ, because these devices need to access a higher layer for remote logins. Figure 3.2 shows the in the last paragraphs explained idealized layer access of various devices [13].

3.2 TCP/IP Model

The TCP/IP model is nowadays the most standard protocol suite used for communication on the internet. It has been the more dominant protocols suit compared to the OSI model and others. The TCP/IP model is not to be confused with the transport protocol TCP which can be used independently from the protocol suite. TCP/IP is a protocol suite that contains multiple protocols, including TCP itself, while TCP is a protocol responsible for building a stable communication channel between two devices for data transfer (see Section 2.1.2). The transport protocol UDP could be used to establish a connection in the TCP/IP protocol suite, which means TCP is not even used in a TCP/IP model. TCP is also used as a layer four protocol in the OSI protocol suite [2] [13].

In the 1970s there were many packet-switching technologies used to communicate

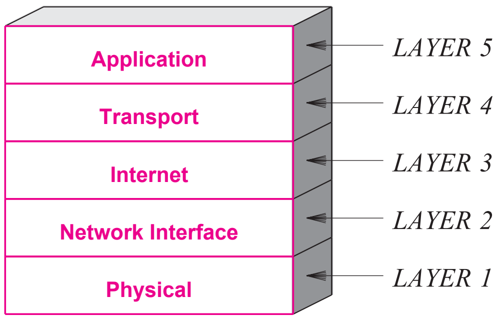

Figure 3.3: The five-layer version of the TCP/IP model
[2]

through the, at the time, archaic internet. The problem with these technologies was that none of them satisfied all requirements for packet switching. So the decision was made to interconnect these technologies into a protocol suite. The created protocol suite is the TCP/IP protocol suite. The name TCP/IP comes from the fact that TCP is the most used transport protocol, that handles packet losses and congestions, on the internet and IP is the dominant protocol used for addressing devices on the internet, so that packets can be sent unambiguously from the sender to receiver (See Section 2.1.2 for more information) [3].

The TCP/IP model is comprised of five layers, which are shown in Figure 3.3. It has two layers less than the OSI model. The five layers combined are also known as a TCP/IP stack. Layers five and four contain protocols used in the host devices, such as the sender and receiver. Layers three to one contain the protocols used by the intermediate devices in an internet connection [2] [13].

The naming of the layers can differ, because some literature draw parallels to the OSI model. As can the number of layers. Some literature have the five-layer model (see Figure 3.3) while others have a four-layer model, which combines the network interface layer/data link layer and the physical layer into one layer called the network access layer [13].

A comparison between the OSI and TCP/IP models is not completely straightforward. There are overlaps of certain protocols of layers. It can roughly be said that layer seven to layer five, the application, presentation and session layer, of the OSI model are contained in the fifth layer, the application layer, in the TCP/IP layer. The rest of the TCP/IP layers are similar to their OSI model counterparts. A good representation of the overlap of the layers of TCP/IP and OSI can be seen in Figure 3.4. But in summary, all five layers of the TCP/IP model combined execute the same tasks as the seven layers of the OSI model [14] [16].

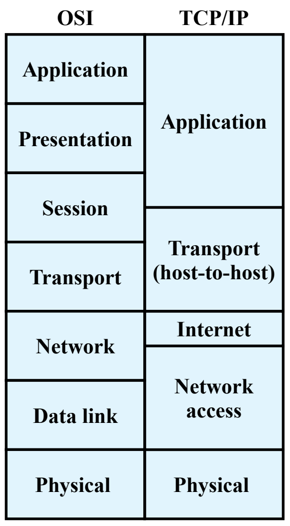

Figure 3.4: Comparing the layers of TCP/IP and OSI model [16]

Chapter 4

MQTT

When it comes to communication between two devices the IP address of the receiving device has to be known. This can lead to difficulties because most devices have temporary IP addresses (see Section 2.1.1). Especially devices that are portable and can connect to the internet through different Networks, for example, a laptop using a public network [3].

To circumvent this problem cloud applications are used. This allows communication between two or more devices through the implementation of a third party. The third party has a fixed IP address that the sender and receiver can reference to receive data. Figure 4.1 shows a laptop communicating with an Internet of Things device through a server of a cloud application vendor. The two communicating end devices have temporary IP addresses, which contact the permanent IP address of the server [3].

An example of such a service is the MQ telemetry transport (MQTT). MQ, often falsely assumed to stand for Message Queuing, stands for a series of products developed by International Business Machines (IBM). MQTT is a lightweight messaging transport protocol. It uses very little bandwidth to send and receive messages, is often utilized in Machine-to-Machine (M2M) communication and is widely seen as easy to implement. The following section will look at how MQTT can be utilised to transport data from one device to another [17] [18].

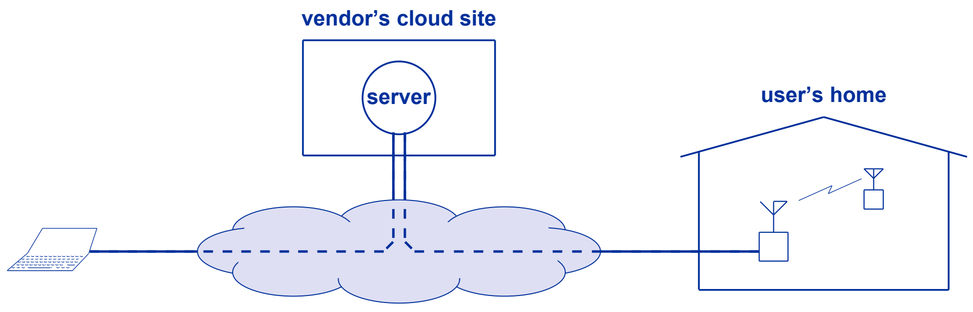

Figure 4.1: Communication between a portable device and an Internet of Things (IoT) device through a cloud application
[3]

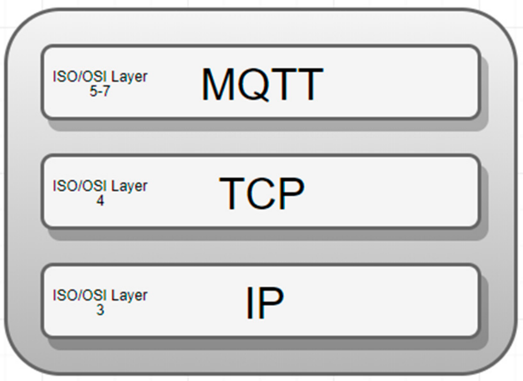

Figure 4.2: MQTT protocols in reference to the layers of the OSI model [19]

4.1 MQTT Architecture

MQTT is based on TCP/IP. Most communications are executed on top of the TCP transport protocol. The implemented protocols in reference to the OSI model layers can be seen in Figure 4.2. It works, compared to the traditional client-server architecture (see Section 2.1.1), on a broker based, publish-subscribe mechanism, also known as pub/sub. In a pub/sub communication, the sender and receiver are decoupled. They do not have direct contact with one another during communication, compared to the client-server architecture [17] [18] [19].

A complete communication between sender and receiver requires a minimum of three entities. These three are the broker and two clients. The broker is the server in between the two clients that manages the messages. It usually has a permanent IP address or one that is known to the clients, so that it always can be connected with. The server of the broker can be from a commercial vendor, for example, HiveMQ, or be set up on a device that is active during the communication sessions. The two clients are split up into publisher and subscriber, these devices do not need a permanent IP address to communicate through MQTT. The publisher is the sender and the subscriber is the receiver of the message [17] [18].

There can be multiple clients connected to the broker. To filter which message goes to which subscriber topics are used. The publisher publishes his message under a topic to the broker. The subscriber can subscribe to this topic on the broker to receive new messages when the publisher sends these to the broker. This means that one message from a publisher can be sent to multiple subscribers if they are subscribed to this topic. One subscribing client can subscribe to multiple topics to receive multiple messages from different, or the same, publishers if he publishes messages in multiple topics, publishers. The broker is responsible for forwarding the messages from the publisher to the right subscriber using the topic. Figure 4.3 shows a publisher client that sends a message under the topic temperature to the broker server, which in this case is a HiveMQ server. Two

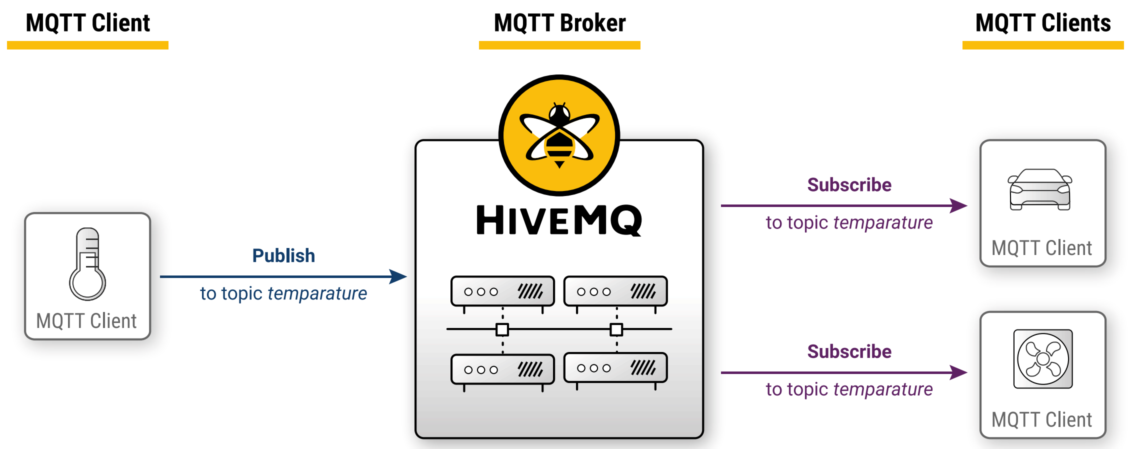

Figure 4.3: Communication between publisher and subscriber through HiveMQ server using MQTT

[17]

subscribed clients receive the message from the broker because both are subscribed to the topic temperature [17] [18].

4.2 Communication of MQTT

An MQTT connection is always initiated by the client, publisher and subscriber. This initialisation starts with a CONNECT message sent by a client to the broker. After receiving this message the broker sends a CONNACK (connection acknowledgment) message back. This sending and receiving of these opening messages can be seen in Figure 4.4. It is to be noted that usually both messages are sent using TCP. This means that there is a guarantee that the message is received when sent (see Chapter 2 for reasoning). After establishing the connection messages can be sent and received bidirectionally by the client and broker, until the client disconnects [17] [18].

After the connection establishment, the client can publish data, subscribe or unsubscribe. Note that a client can be a publisher and subscriber at the same time. Data is published in the payload of a message. The message also specifies for which topic the data is meant for. Subscribing and unsubscribing are accomplished similarly to the connection establishment. The client sends a SUBSCRIBE message, stating the topic to which it wants to subscribe to, to the broker. The broker sends a SUBACK message back acknowledging the subscription. In an analogical manner, the client sends and receives an UNSUBSCRIBE and UNSUBACK message from the broker for unsubscribing from a topic [17].

Another attribute of messages sent to and from the broker is the Quality of Service (QoS). There are three QoS ranging from low to high quality from zero to two. QoS can be different for publishers and subscribers for the same topic. For example, the publisher

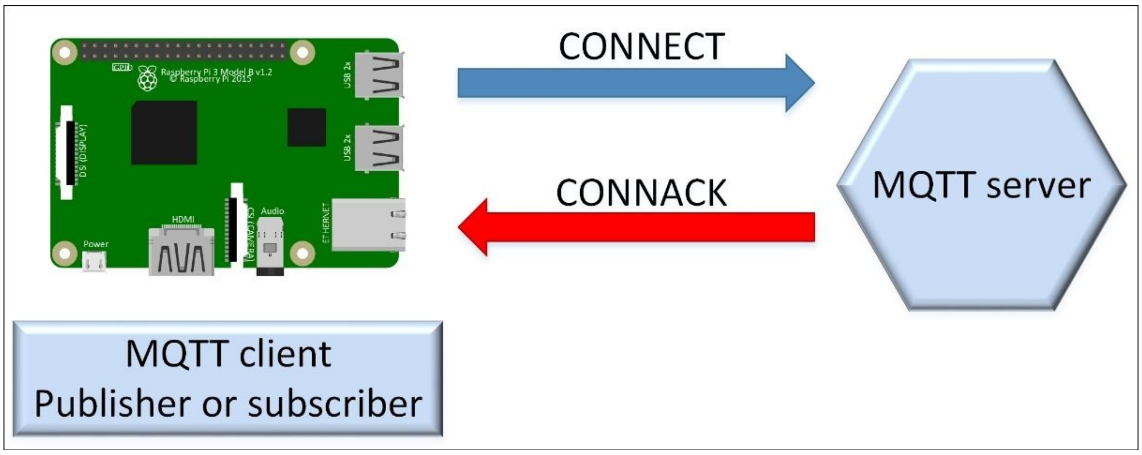

Figure 4.4: CONNECT and CONNACK message being sent between the client and broker

[18]

can send a message for a topic to the broker with QoS two, but the subscriber receives the message of that topic with QoS one. This is an example of a downgrade of QoS. The publisher state the QoS of a message during publication. The subscriber specifies the QoS of the message during his SUBSCRIBE message of a topic [17].

QoS zero, also known as ”fire and forget”, sends a message at most once. This means that there is no guarantee that the message will be delivered to the recipient. QoS zero has the least impact on bandwidth use and is the fastest of the three QoS’ because it is only one single message that is being sent [17].

QoS one sends a message at least once. The receiver is guaranteed to receive the sent message at least once. The sender holds on to the message after sending it until it receives an acknowledgement message, known as PUBACK, from the receiver. If there is no PUBACK message from the receiver within a certain time limit, the sender resends the message waiting again for a PUBACK message. It is possible that the receiver can receive duplicates of the same message [17].

QoS two sends a message exactly once. The receiver gets the message only one time and it is guaranteed to arrive at the receiving end. This is done by a four-part handshake. The sender and receiver use the packet identifier of the published message, which is unique for every published message. First, the sender sends the message with the QoS two to the receiver, while still not discarding the payload of the message. When the receiver receives the message it answers with a PUBREC message to the sender. If the sender receives the PUBREC message it discards the payload of the message, saves the PUBREC message from the receiver and sends a PUBREL message to the receiver. If the sender does not receive a PUBREC message within a certain time limit the message is resent. After the receiver gets the PUBREL message it answers with a PUBCOMP message. Until the receiver sends the PUBCOMP message the identifier of the original sent message is saved so that the receiver can identify if it receives the same packet twice.

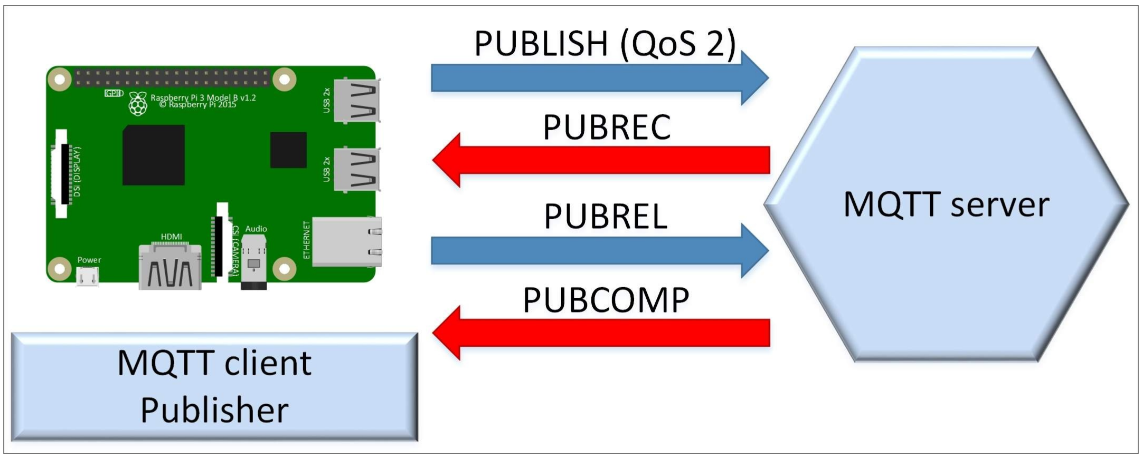

Figure 4.5: Messages sent by publisher and broker for a QoS two message [18]

The four-part handshake is complete when the PUBCOMP message arrives at the sender and the system goes back to the idle condition to send new messages. Figure 4.5 shows a representation of the communication of a QoS two message between the publisher and broker. This QoS is the slowest of the three QoS' [17].

Chapter 5

Controller and Luenberger Observer

A controller is part of a control system, which is a system that outputs a desired result in spite of disturbances. A part of this system is the process, it outputs the value that should be controlled. The actuator is the part of the system that can influence the process. Process and actuators together define the so-called plant. The device that regulates the plant to output the desired value is the controller and will be the focus of this chapter [20] [21] [22].

The main two types of control systems are open-loop and closed-loop. The system is called a closed-loop control system if the plant output is measured with a sensor and fed back into the controller to optimise the plant output. Feedback control systems are more robust against disturbances. The, through a sensor measured, feedback signal is usually subtracted from the desired output, this is more precisely called a negative feedback, which results in the error. This error is passed on to the controller for further configurations. An example of a closed-loop control system can be seen in Figure 5.1. An open-loop control system controls the plant directly with the actuator without a feedback signal from the plant [20] [21] [22].

Controllers can be analog or digital. A digital controller takes in discrete and quantized input signals. When working with a digital controller, it is necessary to convert the signals from analog to digital, with an analog-to-digital converter (ADC), for the input of the controller and from digital to analog, with a digital-to-analog converter (DAC), for the controller output. A main advantage of digital controllers is that it is possible to perform complex operations with the controller input to regulate the plant in a desired method

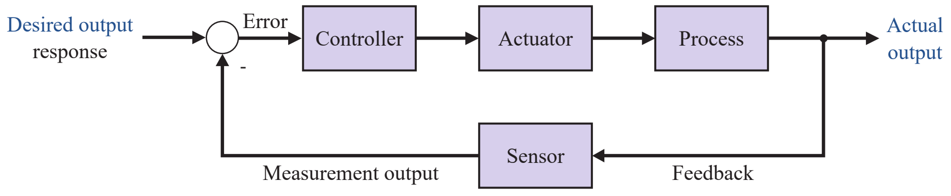

Figure 5.1: Closed-loop control system
[20]

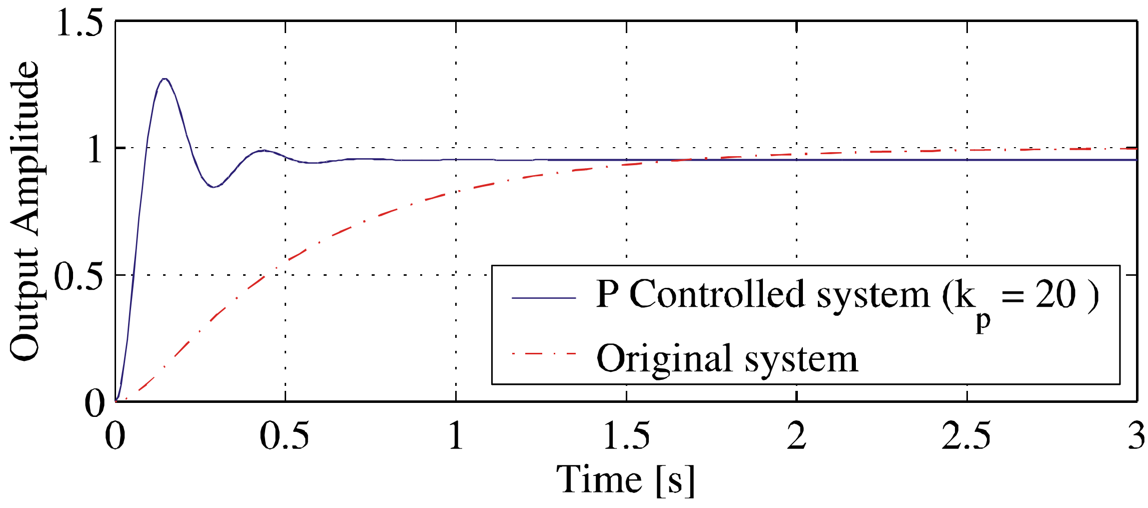

Figure 5.2: Comparison of the original feedback-loop and one with a P Controller [24]

[21].

The Luenberger observer, also known as state observer or just observer, will also be discussed in the last section of this chapter. This observer is used if the assertion of some of the relevant system states is difficult [20].

5.1 PID-Controller

The Proportional Integral and Derivative (PID) Controller is the most used industrial controller. It is a closed-controller usually used to regulate flow, temperature, speed, pressure and other controllable variables. The controller is built up of three terms, whereby it can be built up of only some of the terms. For example the proportional (P) controller, the proportion and Integral (PI) controller, the proportional derivative (PD) controller and with all three terms the PID controller [23] [21].

The proportional part alone can be used as a controller called the P Controller. The transfer function in the Laplace domain of this controller is [23] [21]:

$$C_P(s) = K_p \tag{5.1}$$

K_p is called the proportional gain. Compared to a feedback-loop system that has no controller the value reaches the desired output faster. An issue with the P Controller is the steady-state error. After the value settles down there is still an offset from the desired output. Another issue can be the overshoot which can be problematic depending on the controlled system. See Figure 5.2 for the mentioned characteristics comparing a feedback-loop system with a P controller with $K_p = 20$ [23] [21].

Adding an integral part to the P Controller results in the PI Controller. This controller has the following transfer function in the Laplace domain [23] [21]:

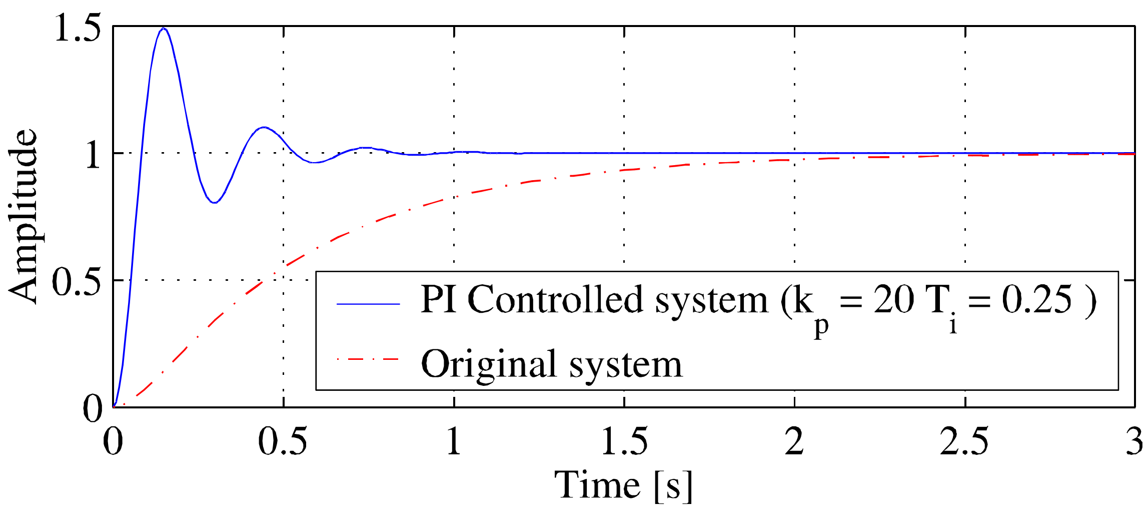

Figure 5.3: Comparison of the original feedback-loop and one with a PI Controller [24]

$$C_{PI}(s) = K_p \left(1 + \frac{1}{T_r s}\right) \tag{5.2}$$

T_r is known as the integral or reset time. The integral term forces the steady state error to zero. A disadvantage of adding the integral term to the controller is that the value takes longer to settle down and the system still overshoots. This can be seen in Figure 5.3. Which shows a controller with $K_p = 20$ and $T_r = 0.25$ [23] [21].

To make the PI Controller faster the derivative term is added to it. The resulting controller is the PID Controller. adding the derivative term also reduces the overshoot of the controlled value. The transfer function in the Laplace domain is [23] [21]:

$$C_{PID}(s) = K_p \left(1 + \frac{1}{T_r s} + T_d s\right) \tag{5.3}$$

T_d is the derivative (action) time. The derivative term is built upon the rate of change of the error. The Figure 5.4 (using $K_p = 20$ $T_r = 0.25$ and $T_d = 0.1$) shows that there is only a slight overshoot and the desired value is reached very fast [23] [21].

Parameters K_p, T_r and T_d can be chosen by different methods. For example the Ziegler-Nichols Method and Reaction Curve Based Method. For further details on these methods refer to [23].

5.2 LQR-Controller

The linear–quadratic regulator (LQR) is an optimal control algorithm. It can be seen as an intermediate control solution between the simpler PID controller and the more complex predictive controllers. It is primarily used on time-invariant feedback systems. A linearisation of the state-space model of a system is required to apply an LQR-Controller.

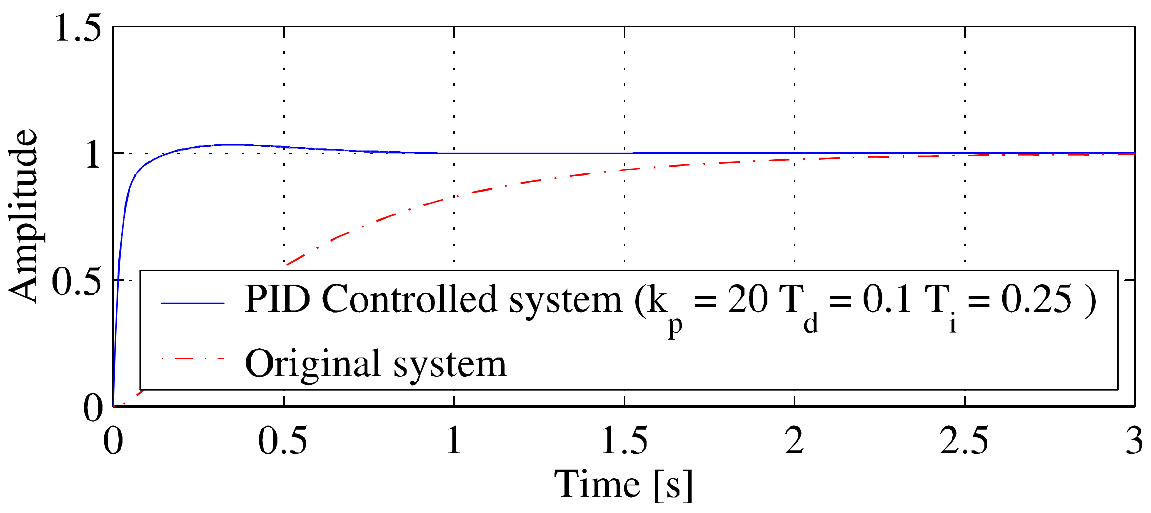

Figure 5.4: Comparison of the original feedback-loop and one with a PID Controller [24]

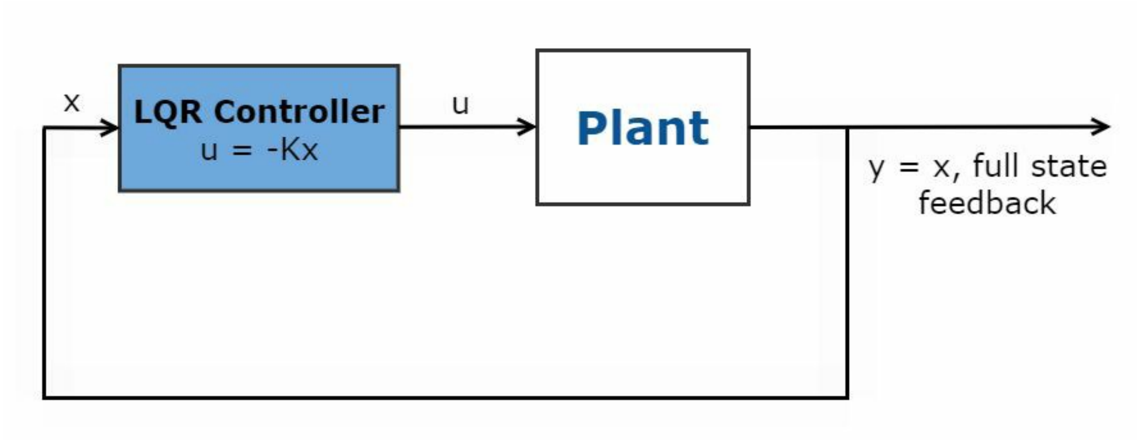

Figure 5.5: State-space model of an LQR controller [27]

Applying the LQR requires more steps to calculate the gain matrix [22] [25] [26].

The LQR-Controller can prioritise change of input values and state values with weighing matrices Q, a positive semi-definite Matrix that ensures system stability, and R, a positive definite matrix that ensures the selected cost is minimized. This makes this controller more intuitive to use. The reasons for prioritization can be that a measured value is more reliable than others or a resource for an input value is more scarce than others [22] [25] [26].

The cost function which the matrices Q and R minimise is:

$$J = \int X^T QX + U^T RU dt \tag{5.4}$$

The advantage of an LQR-Controller is its higher stability. To set up the Q and R matrix good knowledge of the system is required. This Controller is usually calculated numerically and needs more calculation power than a PID Controller [25] [26].

In Figure 5.5 we can see the state-space model of an LQR-Controller. The central task of implementing an LQR-Controller is to calculate the feedback gain of the system. To find this gain first the system matrices A and B have to be known. The weighting matrices Q and R have to be defined depending on the desired system behaviour. One way of calculating the control is by solving the Algebraic Riccati Equation (ARE).

$$A^T S + SA - SBR^{-1}B^T S + Q = 0 \tag{5.5}$$

This gives the solution matrix S. With this matrix we can calculate the full-state feedback gain matrix.

$$K = R^{-1}B^T S \tag{5.6}$$

With this gain, the LQR-controller can be implemented in the system. Which gives the control input of:

$$U = -K \cdot X \tag{5.7}$$

Whereas K is the calculated gain and X is the full state feedback [25] [26].

5.3 Luenberger Observer

In most systems, it is not possible to know the full state vector of the system. So it is difficult to implement a full-state feedback system. Through the combination of the observable outputs and given inputs, it is feasible to estimate the missing parts of the state vector of the system. One way to do this is the Luenberger observer. We will take a closer look at this observer in this section [28] [29].

The goal of the Luenberger observer is to give a result back that is as close as possible to the actual state vector of the system. It does this by using a model of the system and the input and output of this system. See Figure 5.6 for a block diagram of a feedback system with an Luenberger observer. Here the observer takes in the plant input u and output y and calculates the estimated full state vector $\hat{x}$, which is passed on to the Controller [30].

The estimated full state vector for the system

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Dx \end{aligned} \tag{5.8}$$

is given by

$$\hat{x} = A\hat{x} + Bu + L(y - C\hat{x}) \tag{5.9}$$

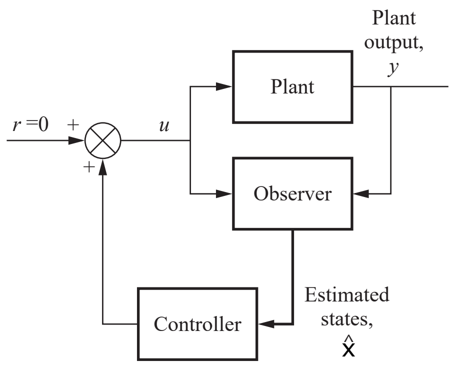

Figure 5.6: Luenberger observer in a full state feedback system with reference value zero [30]

Whereas A, B and C are matrices given by the system. The matrix L is the observer gain that tunes the observer. A disadvantage of using an observer is that poles are added to the overall system for the observer matrix [20] [28].

Chapter 6

Networked Control Systems

A control system with a feedback loop whose components, such as sensor, actuator, plant and controller, are connected over a network, for example, the internet, is considered a Networked Control System (NCS). A typical composition of an NCS, with multiple control systems connected to a network, is shown in Figure 6.1 [12].

The main advantage of an NCS is the possibility to have large physical distances between entities of the control system. It is possible for the controller of the system to be far away from the plant that it controls, as long as it is connected through the network. It is also possible to reconfigure the system during run time. Diagnosis and maintenance of the system are easier and the control system agility is improved. A larger distance can also be desired for reasons of safety. An example would be teleoperations for space projects or nuclear reactor power plant. An NCS also allows for multiple control loops for one or many systems through the network [12] [32] [33] [31].

Utilizing a control system through a network introduces difficulties related to networking because this network is vast and can be populated with other traffic and unknown middleware (see also Section 2.3.1). These problems are, for example, packet losses, safety issues and delays. This can lead to instability in the system and can have detrimental

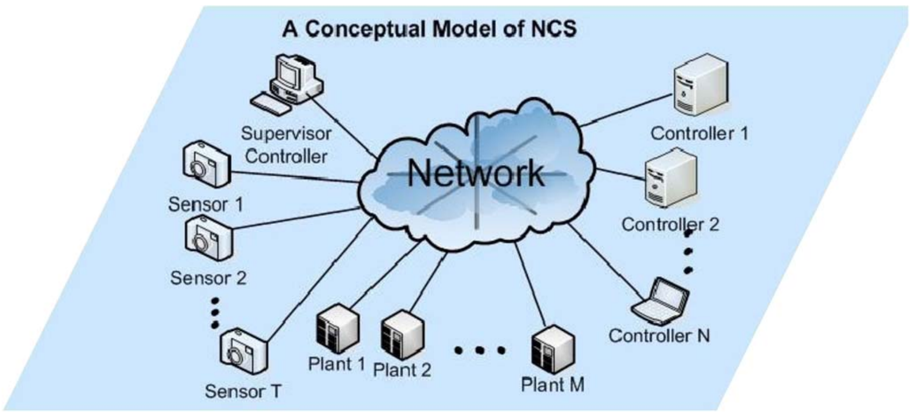

Figure 6.1: Connection of NCS components through a Network [31]

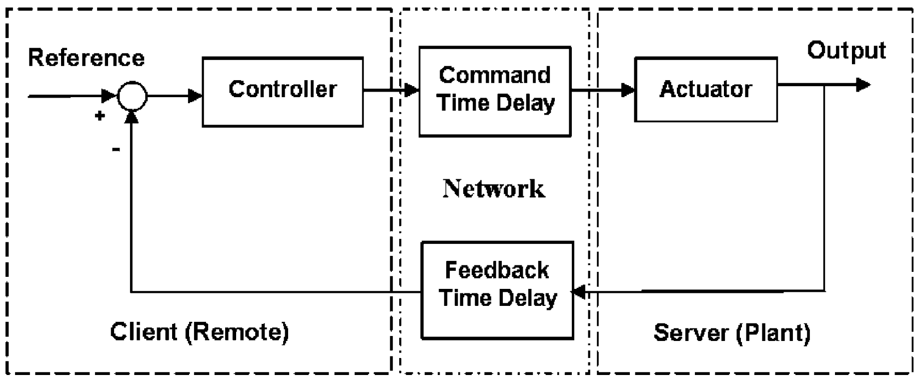

Figure 6.2: Positions of command time delay and feedback time delay in an NCS [33]

effects on it [32] [33] [31].

One of the first NCSs was implemented in the 1950s. It was the flight control system called fly-by-wire. The goal of this innovation was to reduce the fragility, complexity and weight of the at the time used heavy hydromechanical flight control system by introducing electrical circuits as a communication medium in the controlled system. This first analog form of fly-by-wire was fitted in the strategic bomber of the Royal Air Force called Avro Vulcan. The first digital form of the fly-by-wire was used in the modified National Aeronautics and Space Administration (NASA) F-8C Crusader in the year 1972 [31].

Nowadays the NCS is used in many different applications. For example, intelligent traffic control systems, like platooning, which is a control system that regulates the distance between transport vehicles so that all reach their destination safely and fast, automatic warehouse systems, smart power grids and so on. These are realised through wiring (or wireless for example wireless LAN) connection through a network, which can be privately owned or public like the internet, between the controller and the plant. The infrastructure provided by the internet was a boon to NCSs. With the existing Ethernet structure, a network could be connected at a low cost and over a huge area. The wireless NCS (WNCS) is used for mobile systems and is a fast and easy connection method for plant and controller [31].

One of the main reasons for degraded functions and instability of NCS is communication delay. Because data has to travel large distances to arrive at the control system element, it arrives with a time delay. This type of delay is usually seen as constant, whereas the NCS can also experience non-constant delays that vary with time. One of the reasons for these time-invariant delays can be an overload of the network. For further reasons for delays see Section 2.3. Delays in the NCS can be viewed as command time delay, if the delay occurs during data transmission from the controller to the plant, or as Feedback time delay, if the delay occurs during data transmission from the plant to the controller (see Figure 6.2) [33].

These network-induced delays can also be grouped depending on their frequency of occurrence into three main categories [34]:

- Constant delays which have a fixed duration.
- Random delays whose distribution can be modelled by a Markov chain.
- Independent random delays which do not follow any specific distribution.

Control models that are typically used for delay contaminated NCSs are the Proportional Integral Derivative (PID) controller (see also Section 5.1), Model Predictive Control (MPC) and Smith predictor. It is expected that the currently still improving global internet architecture will result in delays becoming less prevalent and shorter [34] [35].

Chapter 7

Measurements of Delay

To observe the delays through the internet, time measurements have been conducted to an MQTT server (see Chapter 4 for more information about the basics of MQTT) of HiveMQ and a Raspberry Pi device at the laboratory of the automation institute of the university of Leoben, from various locations around the world. HiveMQ and the Raspberry Pi serve as brokers for the MQTT setup.

7.1 Setup and Execution

With the help of the Python code in Listing 7.1 the delay time to publish a message, noted as delay in the code, and the delay time to receive the sent message as a subscriber, noted as notification_time in the code, are measured. In this MQTT setup, the brokers each are HiveMQ and the Raspberry Pi device at the University of Leoben. For one measurement session either the HiveMQ or the Raspberry Pi broker is used. Measurement sessions are never executed simultaneously, there is always only one session active at a time, which is communicating with one broker. The publisher and the subscriber are the same device. It is a laptop which is running the Python code in Listing 7.1 (see the Appendix A for the script of the module logger).

Listing 7.1: Python Code used for measuring delays around the world with timer

```
"""DelayMeasurementScript

Purpose: This script measures delay time for a publisher to
publish a message and the the delay time it takes for the
same device to publish and receive a messagethrough MQTT.
The results are documented in a .csv file.

Use (syntax):
    (In command line)
    python3 DelayMeasurementScript LogFileName

Input parameters:
    LogFileName: Enter desired log-file name as an argument in
    the command-line.

Return Parameters:
    Returns a .csv file with the defined log-file name.

This skript uses the client and publish commands from the paho.mqtt
module. Install the paho module within your python enviroment.

The Script also uses the custom made Logger command from the logger
module.

References :

Author :  Gerhard Rath and Gemith Mattathil
Date :    10. January 2022
Version : 3.0

"""

import paho.mqtt.client as mqtt
import paho.mqtt.publish as publish
from time import sleep
import time
import sys
from logger import Logger
import datetime



while True:
  # Getting current time.
  now = datetime.datetime.now().time()

  # Programm starts when time is 10:40.
  if now.hour >= 10 and now.minute >= 40:

    # Checking if a desired log-file name is entered.
    if len (sys.argv) != 2:
      print ("""Needs argument: name of logfile!
              """)
      sys.exit ()
    logfilename = sys.argv[1]
```

```
#================  experiment parameters ================
# Setting up the experimental parameters.
numberOfMeasurements = 360
pauseTime            = 10
commentLine = 'MQTT response test: delay to end of \
  subscription, delay to end of notification'

#==================  subscriber ============
# Can define any server that runs MQTT here by giving
# its IP address or domain name.
# In this case the domain name of the HiveMQ server is used.
# HiveMQ server in Bavaria Germany.
MQTT_SERVER =  "broker.hivemq.com"
MQTT_TOPIC = "mul_iot_ws22"    # Defining the topic.

# Setting up measured parameters.
start = 0.
notification_time = 0.

# The callback function for when the client receives a
# CONNACK response from the server.
def my_connect(client, userdata, flags, rc):
  print("Connected with result code "+str(rc))

  # Subscribing in on_connect() means that if we lose the
  # connection and reconnect then subscriptions will be renewed.
  client.subscribe(MQTT_TOPIC, qos=0)

# The callback function for when a PUBLISH message is
# received from the server.
def my_message(client, userdata, msg):
  global notification_time
  # The payload is the recorded time before publishing.
  result = time.time () - float(str(msg.payload.decode("utf-8")))
  # Result is time taken to publish and receive the msg.payload.
  # In other word this is the Round Trip Time (RTT).
  print(msg.topic+" ", result)
  notification_time = result

# Seeting up the client (with Topic, message and
# subscribing through my_connect).
client = mqtt.Client()
client.on_connect = my_connect
client.on_message = my_message

client.connect(MQTT_SERVER, port=1883, keepalive=60)
# Side-Note: The used port is a TCP connection.
```

```
    # See HiveMQ documentation for further information.

    # Starting a loop for the connection.
    client.loop_start ()

    # Setting up the Logger command with its Parameters.
    out = Logger (logfilename)
    out.setMaxlen(1000000)
    out.log (commentLine) # Setting the first line of the .csv file
    sleep(1)

    # Publishing from this device the time recorded just before
    # publishment in a for-loop that is designed to stop when the
    # required number of measurments is reached.
    for i in range (0, numberOfMeasurements):
      start = time.time () # first recorded time

      # Publishing recorded time as payload to the server under
      # the defined topic with Quality of Service (QoS) equal
      # to 0 (fire and forget).
      publish.single(MQTT_TOPIC, str(start),
                      qos =0, hostname=MQTT_SERVER)

      # delay is time it took to publish (is actually the
      # end-to-end delay, because the connection is built upon TCP).
      delay = time.time () - start

      print (delay)

      # Saving recorded delay (end-to-end delay) and
      # notification_time (RTT) in the defined .csv file.
      out.log (str(delay)+', '+str(notification_time))

      # waiting the defined pauseTime to publish the next message.
      sleep (pauseTime)
    sys.exit()
```

This device measures the delay time from different locations. The measuring period is at least between 10:40 and 15:40, making a minimum of a five-hour measurement duration. The location from which the measurements have been conducted, date of measurement, used MQTT broker and number of measurements are listed in Table 7.1.

The delay parameter can be seen as the end-to-end delay and the notification_time is equivalent to RTT (see also 2.4).

The recorded data was evaluated using Matlab. The subscription delay (end-to-end delay) and notification delay (RTT) are extracted from the .csv file and represented in a histogram. The Matlab code is shown in Listing 7.2.

City	Country	Date	MQTT Server	Number of measurements
Basel	Switzerland	01.06.2022	Raspberry Pi	1830
Chicago	USA	15.06.2022	HiveMQ	1873
		07.06.2022	Raspberry Pi	3660
		16.06.2022	Raspberry Pi	1146
Sendai	Japan	22.08.2022	HiveMQ	2278
		25.08.2022	Raspberry Pi	3023
Leoben	Austria	28.05.2022	HiveMQ	1800
		26.05.2022	Raspberry Pi	1830
Skopje	North Macedonia	24.04.2022	HiveMQ	1744

Table 7.1: Details of the conducted measurements

Listing 7.2: Matlab code to represent the delays as histograms

```
%% Presenting Histogram Data
% Presenting the recorded data of the delays from sendai
   Japan to the HiveMQ
% server in histograms.

% Author : Gemith Mattathil
% Date : 16. March 2022
% Version : 2.0
%% Preping workspace

clear
close all
%% Importing and formating Data
% Importing and formating data from the .csv file into a
   table. Saving the subsription
% delay (end-to-end delay) into the variable 'Delay to end
   of subscription' and
% the notification delay (RTT) into the variable 'Delay to
   end of notification'.
% Also saving the date and time respective variables.
   Lastly converting data format
% from string to double.

RawData = readtable('JapanHive.txt');
RawData.Properties.VariableNames = {'Date','Time','Delay to
    end of subscription', ...
```

```
     'Delay to end of notification'};

% Cleaning up: turning string into double
RawData.("Delay to end of subscription") =  ...
    str2double(RawData.("Delay to end of subscription"));

%% Drawing Histogram
% Drawing the extracted data into histograms two seperate
   histograms for both
% delays.

DelaySub = RawData.("Delay to end of subscription");
DelayNot = RawData.("Delay to end of notification");

% Drawing histogram of delay to end of subscription

histogram(DelaySub)
xlabel('Delay [ms]')
ylabel('Counts [/]')
title('Histogram of subscription delay (end-to-end delay)')


% Drawing histogram of delay to end of notification

histogram(DelayNot)
xlabel('Delay [ms]')
ylabel('Counts [/]')
title('Histogram of notification delay (RTT)')
```

7.2 Results

Most of the histograms have the, in Section 2.4 discussed, typical Gamma-like shape with a heavy tail on the right side. The best representation of this distribution was recorded in Sendai, Japan, to the HiveMQ server in Germany. This distribution is shown in Figure 7.1

To see the outliers more precisely see the zoomed-in version of the histogram in Figure 7.2. The in Section 2.4 discussed deterministic and stochastic parts are well visible in this figure. The deterministic part is about at 0,75 s, every delay after this can be seen as stochastic delays that are caused by random events during the connection.

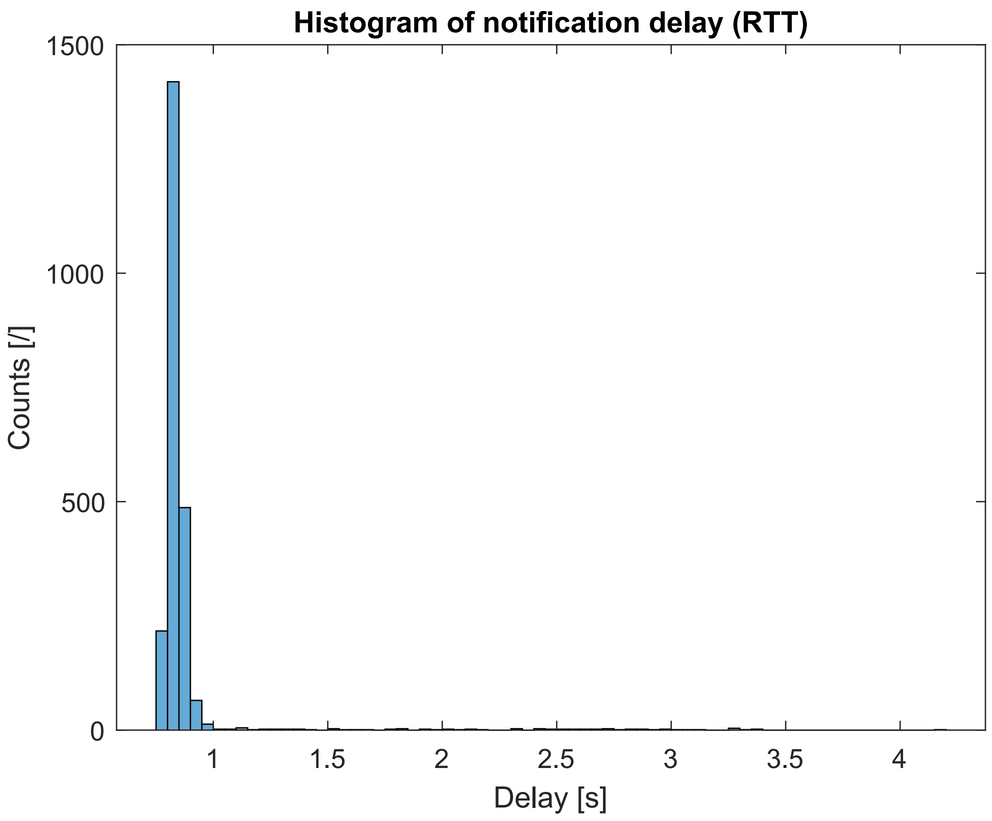

Figure 7.1: Histogram of the notification delays from Sendai, Japan, to HiveMQ server

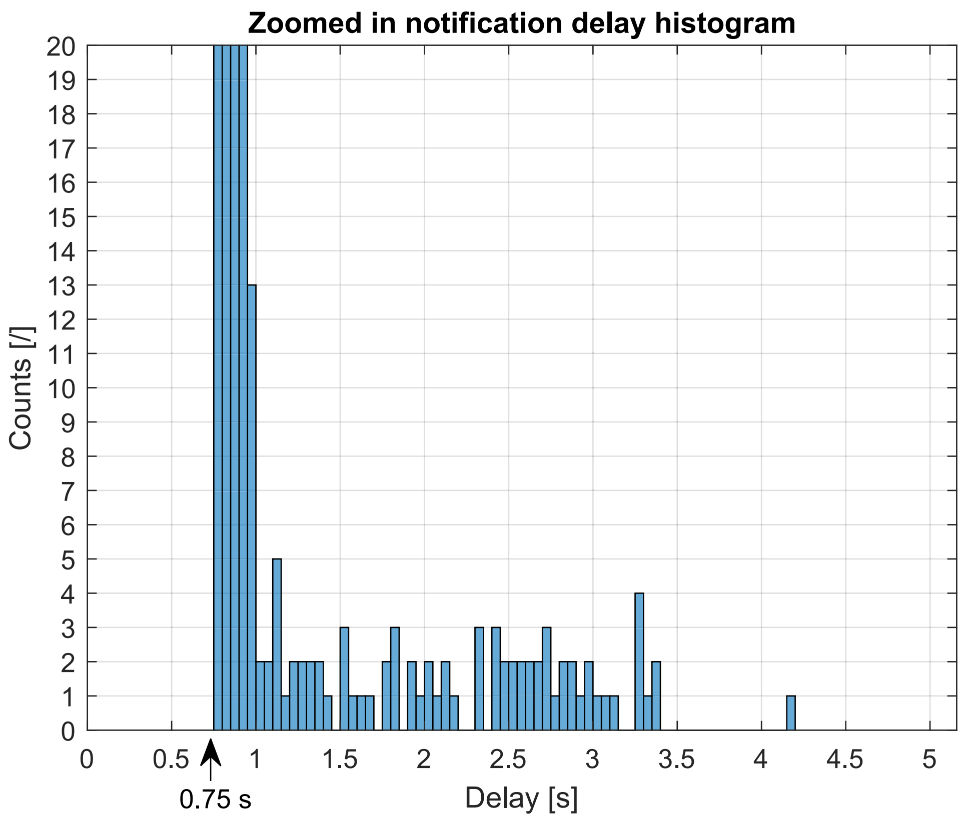

Figure 7.2: Zoomed in Histogram

All other histograms of the measurements can be viewed in Appendix B. The location from which the measurements were conducted and the MQTT server that was used are mentioned under the histograms. All measurements are notification time measurements (RTT).

Chapter 8

Control System With Communication Delay Experiment

8.1 Experimental Setup

The setup for optimising a control system to regulate a second-order system, represented by a circuit with two resistors and two capacitors, despite delays caused by communication through the internet will be shown in this section.

Equipment:

- ESP32
- ESP8266
- Two Capsitors 1000 and 330 μF
- Two Resistors each 1.5 kΩ
- Router (Teltonika RUT240)
- Laptop running Simulink from Mathworks©

The circuit diagram of the system to be controlled is shown in Figure 8.1 it is powered through the ESP32 which is connected to the power supply with a micro USB cable. It is with this ESP that the alternating voltage can be changed. The real-world setup is shown in Figure 8.2.

The Control system is realised through Simulink which is running on a laptop. It gives the control signal to the circuit through the internet.

The communication through the Internet is set up through an MQTT server, the ESP8266, which acts as a bridge, and the router. A separate router (Teltonika RUT240) was used so the traffic could be controlled better during the measurements. The sequence diagram of

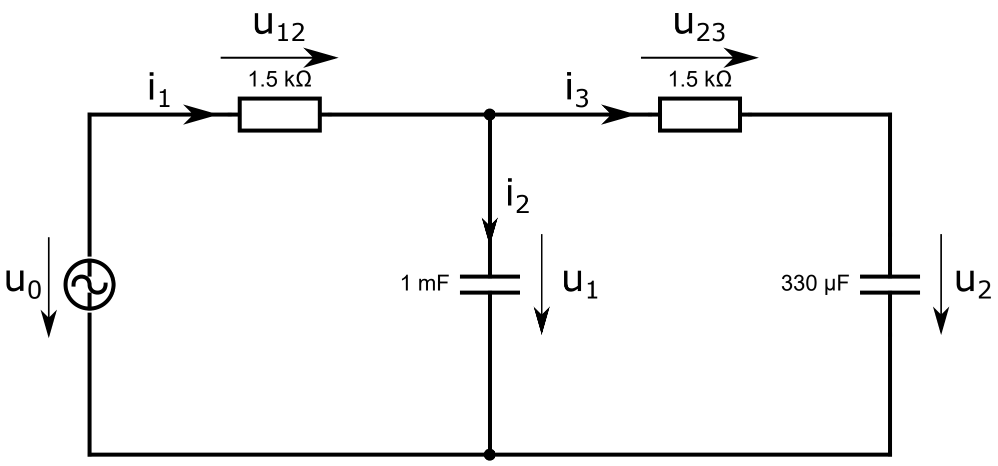

Figure 8.1: Electronic circuit design for experiment

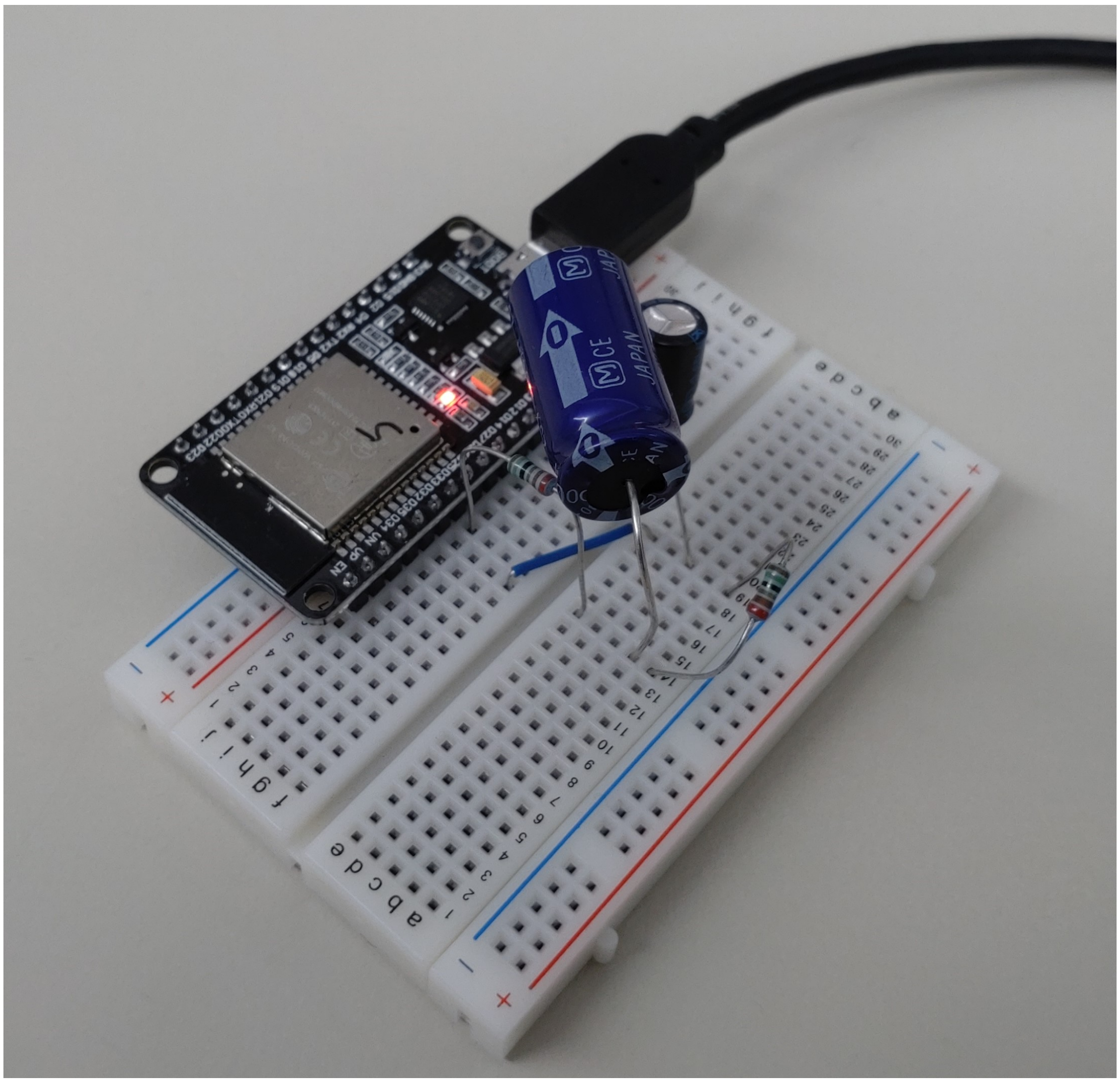

Figure 8.2: Real world circuit

this communication is shown in Figure 8.3. It also includes the type of protocol used to communicate.

First, the controller sends its signal to the ESP8266, which is the bridge or gateway, through UDP. The ESP then relays the signal through TCP to the router and this sends it through the internet to the MQTT server (represented here by HiveMQ). The code for the ESP8266 that acts as the gateway to transform the sent data from UDP to TCP can be seen in Appendix C. After that, the MQTT server sends the information back through the internet to the router, which passes the data to the ESP32. All communication from the Gateway to the ESP32 are executed with TCP. The ESP32 changes the alternating voltage of the circuit according to the controller input. The current value is measured by the ESP32 and sent back to the Router, which forwards it to the MQTT server. The code, written in Python, for the ESP32, can be seen in Appendix E. The server sends the value to the router which relays it to the Gateway and finally to the controller, which is Simulink running on the laptop. The controller calculates the value of the next signal with the received information and the communication cycle starts a new.

To circumvent breaking through the firewall of the university and to have more control over the communication traffic during the measurements, a separate router with a separate internet connection was used (see Figure 8.3).

A direct connection to the HiveMQ server was not possible from the controller, because the controller is programmed in the MATLAB® based graphical programming environment Simulink™ which uses UDP to communicate, while MQTT uses TCP to communicate over the internet. This was the reason for using an ESP8266 as a gateway that transforms UDP into a TCP message (See Figure 8.3).

To see the influences on the results measurements of the communication time were made with and without the gateway. Time was recorded when sending a message from a Raspberry Pi 4 to HiveMQ. Once directly without a gateway through a TCP connection and once indirectly through the ESP8266, which serves as the gateway that transforms UDP into a TCP message. 500 measurements were made on three different days for each type of connection. The 500 measurements were conducted in a row for each type, with and without gateway, of measurement. The averages of the measured time with and without the gateway were calculated and compared. The Python code used for these measurements can be viewed in Appendix D. As can be seen from the results, which are shown in Table 8.1, the influence is smaller than 2 % and can be neglected in the time measurements.

8.2 Calculating State Equations With Bond Graph

For finding the Bond graph for the circuit in Figure 8.1, the elements must be identified. There are two resistors R_1,R_2, (both 1500 Ω) and two capacitors, C_1 (1000 μF)and C_2

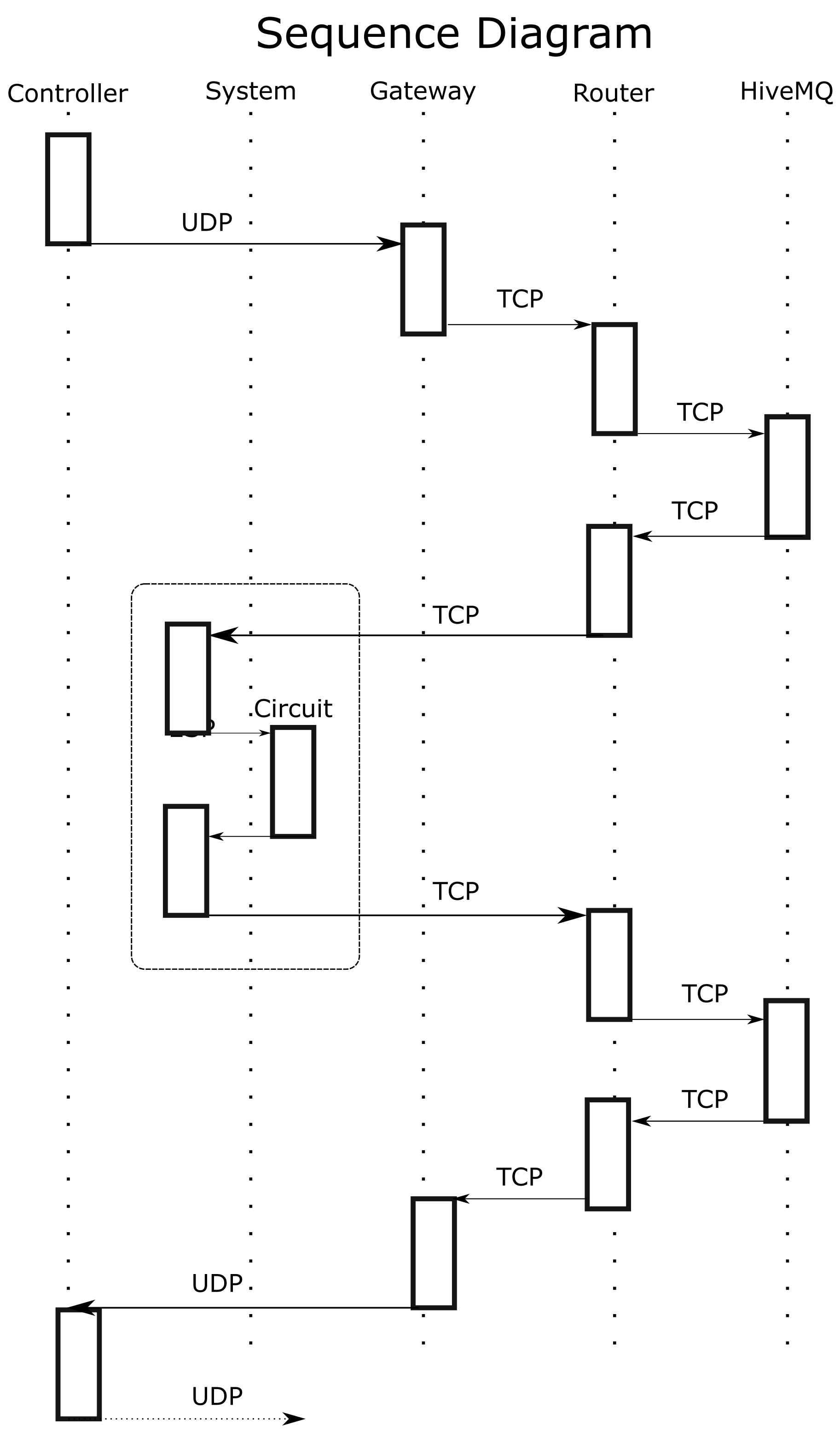

Figure 8.3: Sequence diagram of the communication between the systems

Date & Time	No Gateway [s]	Gateway [s]	Δ [s]	%
22.03.2023 21:00	0.694128372	0.705103911	0.010975539	1.556584628
24.03.2023 11:00	0.69221465	0.705362845	0.013148195	1.864032836
27.03.2023 10:00	0.682304677	0.689144851	0.006840174	0.992559634

Table 8.1: Comparison of time with and without gateway

(330 μF). Another element is the source S_1 of the voltage u_0 [36].

Energy transfer over the bonds is represented by the product of an effort variable and a flow variable, which results in power. For our example of an electrical circuit, flow is electrical current i_i and effort is voltage u_i, of which the product of both is electrical power. For an electrical model, there is a reference effort of zero volts. The other efforts are u_0, which is from a source, u_1 and u_2, which is the output of the process. For each effort except the source, a 0-junction is allocated[36].

The effort differences in the system are $u_{12} = u_0 - u_1$ and $u_{23} = u_1 - u_2$. They are represented by 1-junctions in the graph. All bonds connected to a 1-junction have the same flow variable, which is the current here (Kirchhoff's current law). Then the ports of resistors are connected to the 1-junctions and the ports of capacitors to the 0-junctions. Finally, the flows (currents i_1 to i_4) are identified [36].

The next step is to define the causalities. Causalities define the direction of the computation. They are marked with strokes in the graph, where the stroke represents the direction of the effort flow. One element connected to another via a bond sets the effort variable, while the other element (close to the stroke) defines the flow variable. The source element has a fixed causality. Energy storage elements have preferred causalities. For capacitors the constitutive equation

$$u(t) = \frac{1}{C} \int i \mathrm{d}t. \tag{8.1}$$

is preferred, hence capacity defines the effort variable (voltage). Resistors have indifferent causalities, since the constitutive equation, which is Ohm's law, is algebraic. The remaining causalities in the bond graph are found by propagation [36].

All bonds attached to a 0-junction have the same effort variable, hence only one of them can define the effort. In the actual problem, this is a capacitor defining the voltage. Only one bond attached to a 0-junction is with causality stroke. All bonds attached to a 1-junction have the same flow variable, hence only one of them can define the flow (current). Only one bond attached to a 1-junction is without causality stroke [36].

Propagating the rest of causalities through the graph utilising the indifferent causalities of the resistive elements deliver the final Bond graph in Figure 8.4 [36].

Following are the steps taken to find the differential equations from the derived bond graph.

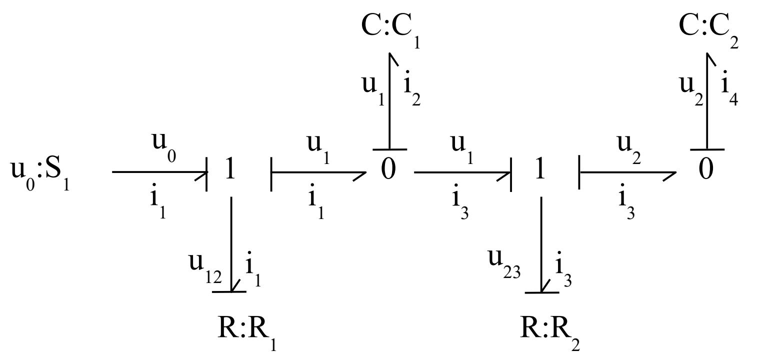

Figure 8.4: Bond graph of the electronic circuit as a system of second order

1-junctions have the same flow variable (current), and the sum of the effort variables (voltage) must be zero [36]. From the graph, in Figure 8.4 follows for the leftmost junction

$$u_0 - u_{12} - u_1 = 0 \tag{8.2}$$

or

$$u_{12} = u_0 - u_1. \tag{8.3}$$

The second 1-junction yields

$$u_1 - u_{23} - u_2 = 0 \tag{8.4}$$

or

$$u_{23} = u_1 - u_2. \tag{8.5}$$

0-junctions have the same effort variable (voltage), and the sum of the flow variables (current) must be zero [36]. For the left 0-junction in Figure 8.4 follows

$$i_1 - i_2 - i_3 = 0 \tag{8.6}$$

or

$$i_2 = i_1 - i_3. \tag{8.7}$$

For the right 0-junction follows

$$i_3 - i_4 = 0 \tag{8.8}$$

or

$$i_4 = i_3. \tag{8.9}$$

The constitutive equations for the capacitive elements are

$$\begin{aligned} \dot{u}_1 &= \frac{1}{C_1} i_2, \\ \dot{u}_2 &= \frac{1}{C_2} i_4. \end{aligned} \tag{8.10}$$

Substituting the currents in (8.10) with the help of (8.7) and (8.9) leads to

$$\begin{aligned}\dot{u}_1 &= \tfrac{1}{C_1}\left(i_1 - i_3\right),\\ \dot{u}_2 &= \tfrac{1}{C_2} i_3.\end{aligned} \tag{8.11}$$

Further, the constitutive equations of the resistive elements (simply Ohm's law) allow to substitute the currents and make

$$\begin{aligned}\dot{u}_1 &= \tfrac{1}{C_1}\left(\tfrac{u_0-u_1}{R1} - \tfrac{u_1-u_2}{R2}\right),\\ \dot{u}_2 &= \tfrac{1}{C_2}\cdot\tfrac{u_1-u_2}{R_2}.\end{aligned} \tag{8.12}$$

Writing (8.12) in matrix form yields directly the state-space representation

$$\begin{bmatrix}\dot{u}_1\\ \dot{u}_2\end{bmatrix} = \begin{bmatrix}-\frac{1}{C_1R_1}-\frac{1}{C_1R_2} & \frac{1}{C_1R_2}\\ \frac{1}{C_2R_2} & -\frac{1}{C_2R_2}\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix} + \begin{bmatrix}\frac{1}{C_1R_1}\\ 0\end{bmatrix} u_0. \tag{8.13}$$

8.3 Simulation

The actual dynamic system is modelled with its state space representation. A Linear time-invariant (LTI) system can be described with the differential equation

$$\dot{\mathbf{x}}(t) = \mathbf{A}\cdot\mathbf{x}(t) + \mathbf{B}\cdot\mathbf{u}(t), \tag{8.14}$$

where $\mathbf{x}$ is the vector of state variables, such as position, speed, and so on. $\mathbf{A}$ is the Dynamic Matrix, $\mathbf{B}$ is the Input Coupling Matrix, and $\mathbf{u}$ is the vector of input variables [21]. In the actual system for experiments Equation 8.13, the state variables are the voltages across the capacitors. The solution of this system of first-order ODEs will be

$$\mathbf{x}(t) = e^{\mathbf{A}t}\cdot\mathbf{x}_0 + \int_0^t e^{\mathbf{A}(t-\tau)}\cdot\mathbf{B}\cdot\mathbf{u}(\tau)\,\mathrm{d}\tau. \tag{8.15}$$

The first part is the solution of the homogeneous system without inputs. It finds the state at a time t from the initial conditions $\mathbf{x_0}$. The second part describes the effect of the input and is a convolution of the matrix exponential with the input vector. In the case of sampling systems, the inputs are assumed as constant values within a certain time interval [21]. Hence one of the variables to be convolved is a constant, which allows an analytical solution. The state variables at a time t_n can be obtained starting from the previous sample at t_{n-1} with

$$\mathbf{x}(t_n) = e^{\mathbf{A}(t_n-t_{n-1})}\cdot\mathbf{x}_{(t_{n-1})} + \left(e^{\mathbf{A}(t_n-t_{n-1})} - \mathbf{I}\right)\cdot\mathbf{A}^{-1}\cdot\mathbf{B}\cdot\mathbf{u}_{t_{n-1}}. \tag{8.16}$$

Usually in digital sampling systems, the time difference is a constant sampling time.

This is not valid for systems subjected to communication delays. As a consequence, it is difficult to use conventional simulation tools, such as Simulink™. For the actual work, simulation is done in MATLAB®. The progress in time is done by advancing from one time instant to the next using stateless functions. For the implementation of the dynamic system in Equation 8.16 the function *process* in Listing 8.1 is used. The simulated system for the experiments is the electronic network from Equation 8.13. The functions contain the limiting of input and output, and furthermore the scaling for the ADC and DAC of the ESP32 system. The variable u is the control input to the process, t_0 is the starting time, and t_1 is the final target time. For a continuous simulation, the function can be invoked within a loop, when the output state x is used as function input x_0 in the next cycle.

Listing 8.1: Matlab code of the function *process* which represents the system

```
function [y, x] = process(A, B, C, x0, u, t0, t1)
    if u < 0
        u = 0;
    elseif u > 511
       u = 511;
    end
    u = u*10.8; % scale to adc and dac
    dim    = length (B);
    expo0 = expm (A*(t1-t0));
    conv0 = (expo0 - eye(dim))*inv(A) * B;
    x = expo0*x0 + conv0*u;
    y = C*x;
    if y < 0
        y = 0;
    elseif y > 4095
        y = 4095;
    end
end
```

The implementation of a PI controller (ref Equation 5.2) is the function *picontro2* shown in the Listing 8.2. The integrator requires an additional state, which must be available for the next call. It is represented by the variable i_0.

Listing 8.2: Matlab code of PI controller as the function picontro2

```
function [u, i0] = picontr2 (Kp, Tr, i0, setval, actval, t0
   , t1)
```

```
    e = setval - actval;
    u = kp*(e+i0);
    i0 = i0 + e*(t1-t0)/Tr;
end
```

When the output of the system is measured, the other state variables are not accessible. With the help of an observer, the function *observer* in the Listing 8.3, the missing states can be estimated. Appendix F shows the Matlab code for the implementation of this. The measurement value is y_0.

Listing 8.3: Matlab code for the function of the observer

```
function [xobs, err] = observer (A, B, C, L, u, xobs, y0,
   y1, t0, t1)

    u = u*10.8; % scale to adc and dac
    dim    = length (B);

    expo0 = expm (A*((t1-t0)/2));
    conv0 = (expo0 - eye(dim))*inv(A) ;
    err0 = y0 - C*xobs;
    xobs = expo0*xobs + conv0*B*u + conv0*L*err0;
    err1 = (y1+y0)/2 - C*xobs;
    xobs = expo0*xobs + conv0*B*u + conv0*L*err1;
    err = y1 - C*xobs;
end
```

An example listing for the system simulation is shown in Appendix F. It displays a PI control applied to the system. Communication with stochastic delays is programmed with a finite-state machine.

8.4 Experiments With the Laboratory Equipment

The equipment for the process model is described in 8.1. The system under control is the electronic network in Figure 8.1, and its state equation used for simulation is Equation 8.13. The microcontroller ESP32 asserts the input voltage (controller output) and reads back the analog output (system response). The connection to the internet is done over a WiFi local network and an LTE router. The communication peer is a standard PC, that exchanges the data over the internet with the MQTT protocol, using HiveMQ as a remote service. The experiments can be done with programs written in Python, MATLAB®,

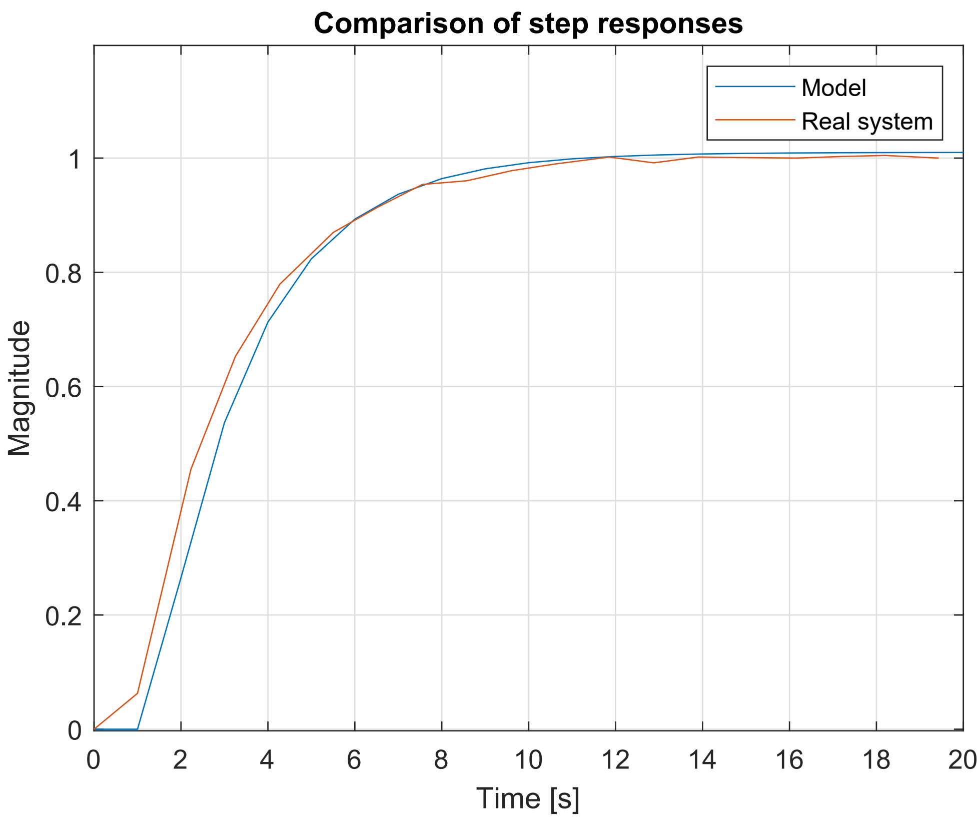

Figure 8.5: Measured and simulated step responses

Simulink™, or even manually using an MQTT plugin in Visual Studio Code (VSC). If the PC is connected to the same WiFi network, the transmission can be done also using the UDP protocol via the ESP8266 gateway. This is necessary when working with Simulink since it cannot handle MQTT at the present time.

8.4.1 Step Reponse

The system's state equation in Equation 8.13 contains the values of the electronic components, which build the actual system under control. A discrete transfer function with a sample time of one second was generated with Matlab, and one additional delay step for modelling the communication is inserted. A Python program was used for doing the measurements and recording the data. Figure 8.5 displays the results.

8.4.2 PI control

An experiment with a controller realised in Simulink™was done. The block diagram is shown in Fig. 8.6.

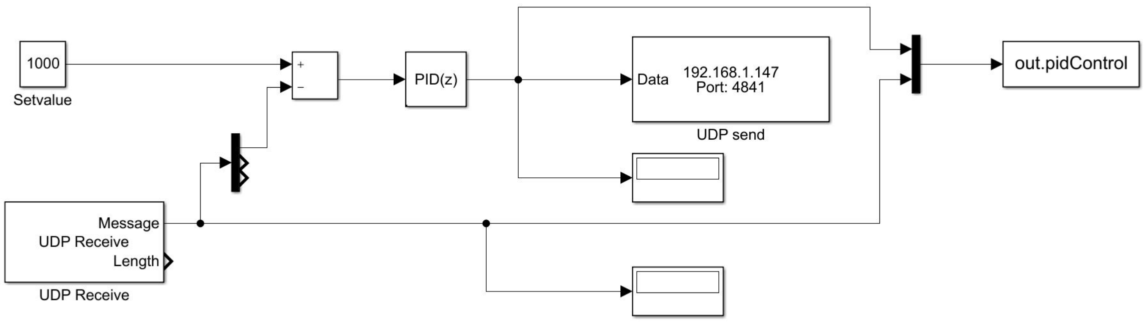

Figure 8.6: PI controller in Simulink

As the sampling time, one second was chosen. The transfer function of a PI controller

$$C_{PI}(s) = K_p \left(1 + \frac{1}{T_r s}\right) \tag{8.17}$$

is applied in its discrete form with the parameters $K_p = 0.1$ and $T_r = 10$. Simulink™ uses for the integrator the approximation

$$I_1(s) = \frac{1}{z-1}, \tag{8.18}$$

whereas the simulator written in Matlab code in Appendix F is based on

$$I_2(s) = \frac{z}{z-1}. \tag{8.19}$$

Experiments showed that the difference has no big influence on the results. In the simulator, the delays for sending and receiving are 0.4 seconds plus a random time with uniform distribution with an amplitude of 0.4 seconds. The comparison of the results is shown in Fig. 8.7. The times plotted as x-axes are the times at the system side, which are transmitted together with the process response. The results are highly dependent on the stochastic delays of both the equipment and the simulation. Only the Round-Trip Time (RTT) can be measured, consequently, there is no possibility to separate sending time from receiving time.

8.4.3 Observer for the Process

The goal of an NCS is to keep the tasks of the remote system as simple as possible and to perform the complex algorithms at the controller site. Even simple microprocessors usually have a precise time base. This can be utilised for an additional benefit. When transmitting only a single sensor value, UDP and TCP have a high overhead. Adding one or a few more data points do not make any difference in the communication. In the actual experimental setup, the remote system sends back the actual value of the process output y_n, of course, but also the control value u_n and its local timestamp t'_n, see Fig. 8.8.

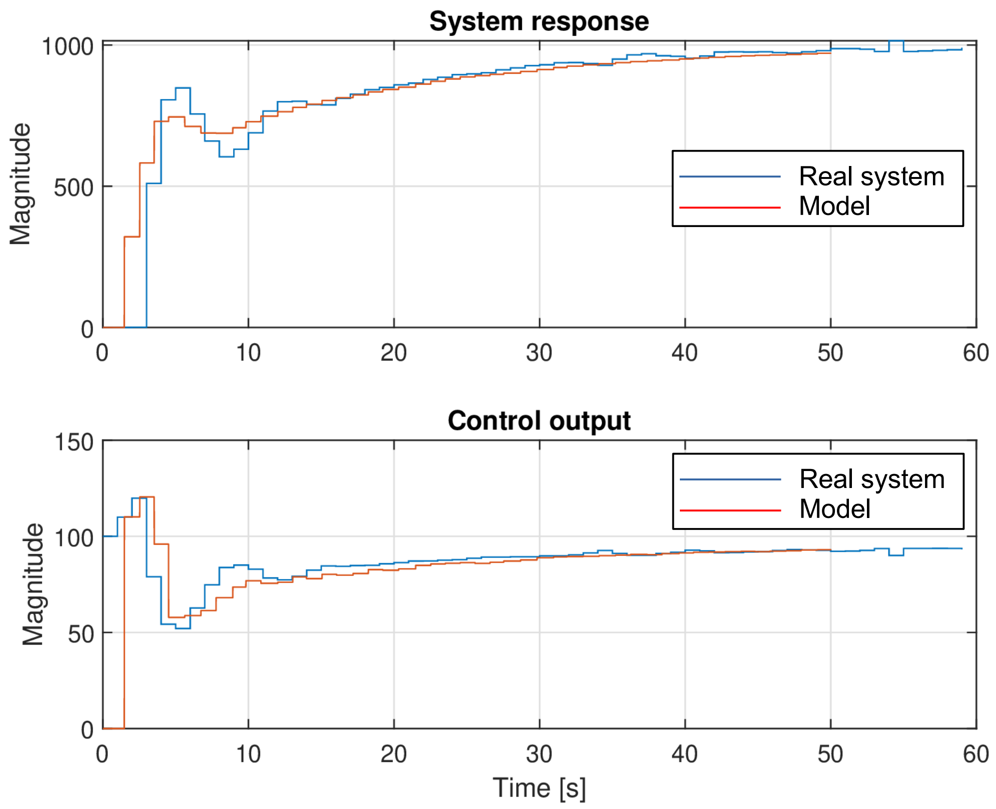

Figure 8.7: Measured and simulated PI control

This configuration allows an identification or observation of the process in the local time scale of the remote system. Important to mention is that the time stamps have no regular intervals. The observer presented in Section 8.3 can handle this. Figure 8.9 shows the response of the observer at the controller site.

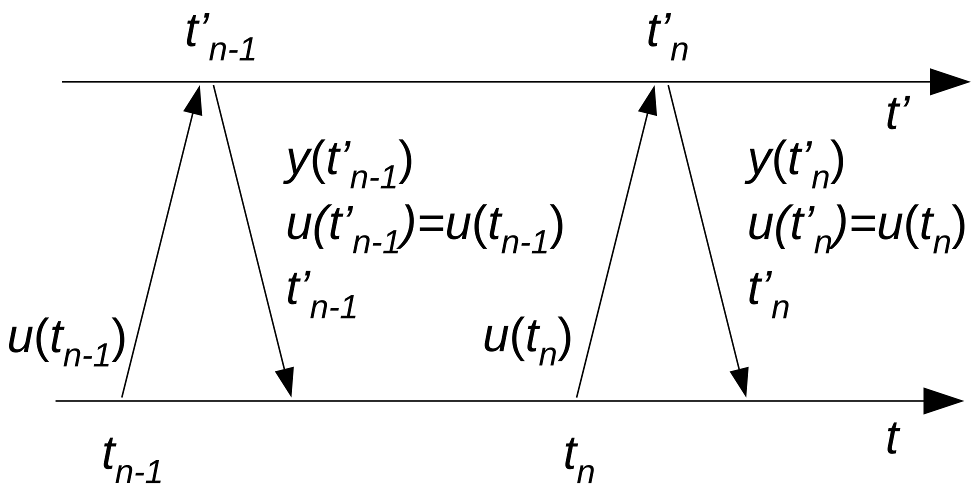

Figure 8.8: Communication sequence

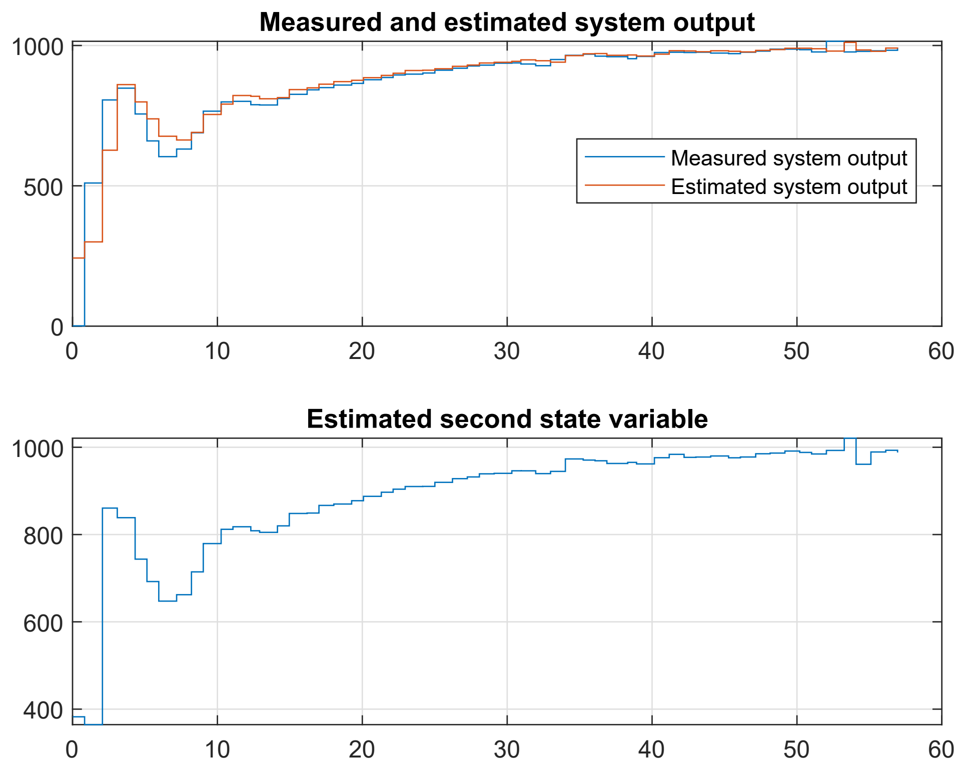

Figure 8.9: Observer output

Chapter 9

Conclusion

Measurements of the round-trip time (RTT) were conducted from different parts of the world to a Raspberry Pi in the laboratory of the university in Leoben, Austria and an MQTT server of HiveMQ in Bavaria, Germany. These measurements were conducted with Python code (see Listing 7.1) on a laptop. The measurement duration was on average about five hours. The countries from which the measurements were conducted were Austria, Switzerland, North Macedonia, USA and Japan.

After collecting the data of the measurements, the results were presented in histograms using Matlab®. These graphically displayed outcomes gave a good overview of the delays in the Internet. It gave an insight into the statistical distribution, more precisely the deterministic and stochastic part, and the duration of delay in communication over a networked system like the internet. As well as the amount of less common disturbances in the form of outliers.

An experimental configuration was assembled for the setup of a process of second order model with an electronic network, driven and measured with an ESP32 microcontroller. The communication of this model through the internet was established with TCP to an external MQTT broker.

The broker relayed the messages from the second order system to a PC. This PC is running a controller which can be programmed in Python, Matlab or Simulink. The communication of the controller was initiated with UDP. An ESP8266 microcontroller was programmed to act as a gateway, which converts UDP, from the controller, to TCP datagrams, for the MQTT broker traffic, and vice versa. It was made sure (see Section 8.1) that the gateway does not have relevant effects on the RTT. The controller and system which communicate with an MQTT broker connected through the internet is as a whole a Network Control System (NCS).

For analysis of such NCS, it may be desired to have reproducible experiments. The system parameters were determined with the help of the bond graph analysis. Missing state vector components were estimated with a Luenberger Observer. Simulations were conducted in Matlab of the physical process including stochastic communication delays for this purpose. In order to address and resolve the issue pertaining to the non-uniform sampling time an effective and viable solution had to be identified. Verifying experiments comparing the results of the physical laboratory system and the output of the simulation were made.

To compensate for the non-uniform sampling time a method is proposed to make accurate observations and identifications of the process. The attainment of this objective is accomplished by enabling the remote system to autonomously regulate its own time scale and subsequently provide the controller with the relevant data.

Table 9.1: List of Abbreviations

Abbreviations	Expansion
MQTT	Message Queuing Telemetry Transport
UDP	User Datagram Protocol
IoT	Internet of Things
CPU	Central Processing Unit
M2M	Machine-to-Machine
LAN	Local Area Network
WAN	Wide Area Network
OSI	Open Systems Interconnection
Wi-Fi	Wireless Fidelity
ISP	Internet Service Provider
IP	Internet Protocol
DHCP	Dynamic Host Configuration Protocol
MAC	Media Access Control
SMTP	Simple Mail Transfer Protocol
HTTP	Hypertext Transfer Protocols
FTP	File Transfer Protocol
TCP	Transmission Control Protocols
Mbps	Megabits per second
Gbps	Gigabits per second
BGP	Border Gateway Protocol
NIC	Network Interface Controller
RTO	Retransmission Timeout
RTT	Round-Trip Time
AIMD	Additive Increase Multiplicative Decrease
ARPANET	Advanced Research Projects Agency Network
ARM	ARPANET Reference Model
ISO	International Organisation for Standardization
ASCII	American Standard Code for Information Interchange
MPEG	Moving Pictures Experts Group
RPC	Remote Procedure Call
QoS	Quality of Service
LTI	Linear Time Invariant
PID	Proportional Integral and Derivative
NCS	Networked Control System
WNCS	Wireless Networked Control System
ADC	Analog-to-Digital Converter
DAC	Digital-to-Analog Converter
NASA	National Aeronautics and Space Administration
Gbps	Gigabits per second
MPC	Model Predictive Control
VSC	Visual Studio Code
LQR	Linear–Quadratic Regulator

Bibliography

[1] D. Liu and F.-Y. Wang, Eds., *Networked Control Systems: Theory and Applications*, ser. SpringerLink Bücher. London: Springer London, 2008.

[2] D. Comer, *Computer Networks and Internets*, 5th ed. Pearson, 2009.

[3] D. Comer, *Internet book: Everything you need to know about computer networking and how the internet works*. Pearson, 2018.

[4] L. Williams, "Ipv4 vs ipv6 – difference between them", 2020. [Online]. Available: `https://www.guru99.com/difference-ipv4-vs-ipv6.html` (visited on 11/02/2022).

[5] A. 19, *How the internet really works: An illustrated guide to protocols, privacy, censorship, and governance*. No Starch Press, 2021.

[6] *17 tcp transport basics — an introduction to computer networks, desktop edition 2.0.9*, 2022. [Online]. Available: `https://www.intronetworks.cs.luc.edu/current2/html/tcpA.html` (visited on 11/04/2022).

[7] G. Rath, G. Probst, and W. Kollment, "Robot control unit for educational purposes", 2015.

[8] S. Bauer, D. D. Clark, and W. Lehr, "Understanding broadband speed measurements", 2010.

[9] L. Angrisani, S. D'Antonio, M. Esposito, and M. Vadursi, "Techniques for available bandwidth measurement in ip networks: A performance comparison", vol. 50, 2006.

[10] N. Feamster and J. Livingood, "Measuring internet speed", *Communications of the ACM*, 2020.

[11] C. Bovy, H. Mertodimedjo, G. Hooghiemstra, H. Uijterwaal, and P. Van Mieghem, "Analysis of end-to-end delay measurements in internet", in *Proc. of the Passive and Active Measurement Workshop-PAM*, vol. 2002, 2002.

[12] A. Bemporad, *Networked Control Systems*. London: Springer London Limited, 2010, vol. v.406.

[13] K. R. W. R. S. Fall, *TCP/IP illustrated*, 2nd ed. Addison-Wesley, 2012.

[14] J. F. Kurose and K. W. Ross, *Computer networking: A top-down approach*, 6. ed. Pearson, 2013.

[15] E. Atxutegi, F. Liberal, E. Saiz, and E. Ibarrola, “Why we still need standardized internet speed measurement mechanisms for end users”, in *Trust in the information society*, Piscataway, NJ: IEEE, 2015.

[16] W. Stallings, *Data and computer communications*, 8. ed. Pearson Prentice Hall, 2007.

[17] HiveMQ, *MQTT & MQTT 5 Essentials: A comprehensive overview of MQTT facts and features for beginners and experts alike*. Landshut germany: HiveMQ, 2020, vol. 1.

[18] G. C. Hillar, *MQTT essentials: A lightweight IoT protocol : the preferred IoT publish-subscribe lightweight messaging protocol*. Birmingham, UK: Packt Publishing, 2017.

[19] C. Toma, A. Alexandru, M. Popa, and A. Zamfiroiu, “Iot solution for smart cities’ pollution monitoring and the security challenges”, *Sensors (Basel, Switzerland)*, vol. 19, 2019.

[20] R. C. Dorf and R. H. Bishop, *Modern control systems*, Fourteenth edition, global edition. Harlow: Pearson, 2022.

[21] G. F. Franklin, D. J. Powell, and M. L. Workman, *Digital control of dynamic systems*, 3. ed. Menlo Park, Calif.: Addison Wesley Longman, 2002.

[22] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, Eds., *Feedback control of dynamic systems*, 7. ed., Global ed., ser. Always learning. Boston, Mass.: Pearson, 2015.

[23] G. C. Goodwin, S. F. Graebe, and M. E. Salgado, *Control system design*. Prentice Hall, 2001.

[24] C. of Automation Montanuniversität Leoben.

[25] P. Saraf, M. Gupta, and A. M. Parimi, “A comparative study between a classical and optimal controller for a quadrotor”, in *2020 IEEE 17th India Council International Conference (INDICON)*, IEEE, 2020.

[26] J. G. Batista, D. A. Souza, L. L. dos Reis, *et al.*, “Performance comparison between the pid and lqr controllers applied to a robotic manipulator joint”, in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, IEEE, 2019.

[27] *What is optimal control?* [Online]. Available: `https://ch.mathworks.com/discovery/optimal-control.html` (visited on 05/21/2023).

[28] D. G. Luenberger, “Observing the state of a linear system”, *IEEE Transactions on Military Electronics*, vol. 8, 1964.

[29] D. Luenberger, “An introduction to observers”, *IEEE Transactions on Automatic Control*, vol. 16, 1971.

[30] N. S. Nise, *Control systems engineering*, Seventh Edition. Hoboken: Wiley, 2015.

[31] R. A. Gupta and M.-Y. Chow, “Networked control system: Overview and research trends”, *IEEE Transactions on Industrial Electronics*, vol. 57, pp. 2527–2535, 2010.

[32] Wei Zhang, M. S. Branicky, and S. M. Phillips, “Stability of networked control systems”, *IEEE Control Systems Magazine*, vol. 21, pp. 84–99, 2001.

[33] C.-L. Lai and P.-L. Hsu, “Design the remote control system with the time-delay estimator and the adaptive smith predictor”, *IEEE Transactions on Industrial Informatics*, vol. 6, pp. 73–80, 2010.

[34] T. C. Yang, H. Yu, Fei, and L. X. Li, “Networked control systems: A historical review and current research topics”, *Measurement and Control*, vol. 38, pp. 12–16, 2005.

[35] H. Xing, J. Ploeg, and H. Nijmeijer, “Smith predictor compensating for vehicle actuator delays in cooperative acc systems”, *IEEE Transactions on Vehicular Technology*, vol. 68, pp. 1106–1115, 2019.

[36] D. C. Karnopp, D. L. Margolis, and R. C. Rosenberg, *System Dynamics: Modeling and Simulation of Mechatronic Systems*, 3rd ed. John Wiley & sons, New York, 2000.

Appendix A

Logger Module

Listing A.1: Python Code for the module used to save the delay data to a .csv file

```python
# Author :  GERHARD RATH
# Date :    8. February 2021
# Version : 3.0

import os
from time import localtime, strftime, time


# to log

class Logger:
  """Manages a logfile in 2 out of 3 rotating parts"""
  def __init__(self, name):
    self.maxlen = 100
    self.path = ''
    self.a = self.b = self.c = False
    self.name = name
    self.aFile = name+'A.txt'
    self.bFile = name+'B.txt'
    self.cFile = name+'C.txt'

  def setPath(self, path):
    self.path = path
  def setMaxlen(self, len):
    self.maxlen = len
  def log (self, message):
    now = time()
    loctime = localtime(now)
    # gives millisec with three digits
    millisecs = '%03d' % int((now - int(now)) * 1000)
    tim = strftime('%Y-%m-%d %H:%M:%S.', loctime) + millisecs + ' '
    self.a = os.path.exists(self.path + self.aFile)
    self.b = os.path.exists(self.path + self.bFile)
    self.c = os.path.exists(self.path + self.cFile)
    if (not self.a and not self.b and not self.c):    # no file
      # If file exists it appends, if not it creates
      f = open(self.path + self.aFile, 'a+')
      f.write(tim + message +'\n')
    elif (self.a and not self.b and not self.c):      # only A
      if os.path.getsize(self.path + self.aFile) < self.maxlen:
        f = open(self.path + self.aFile, 'a+')
      else:
        f = open(self.path + self.bFile, 'a+')
      f.write(tim + message +'\n')
      f.close()
    elif (not self.a and self.b and not self.c):      #only B
      if os.path.getsize(self.path + self.bFile) < self.maxlen:
        f = open(self.path + self.bFile, 'a+')
      else:
        f = open(self.path + self.cFile, 'a+')
      f.write(tim + message +'\n')
      f.close()
    elif (not self.a and not self.b and self.c):      #only C
      if os.path.getsize(self.path + self.cFile) < self.maxlen:
        f = open(self.path + self.cFile, 'a+')
      else:
        f = open(self.path + self.aFile, 'a+')
      f.write(tim + message +'\n')
      f.close()
    elif (self.a and self.b and not self.c):      # A + B
      if os.path.getsize(self.path + self.bFile) < self.maxlen:
        f = open(self.path + self.bFile, 'a+')
      else:
        os.remove(self.path + self.aFile)
        f = open(self.path + self.cFile, 'a+')
      f.write(tim + message +'\n')
      f.close()
    elif (not self.a and self.b and self.c):      # B + C
      if os.path.getsize(self.path + self.cFile) < self.maxlen:
        f = open(self.path + self.cFile, 'a+')
      else:
        os.remove(self.path + self.bFile)
        f = open(self.path + self.aFile, 'a+')
      f.write(tim + message +'\n')
      f.close()
    elif (self.a and not self.b and self.c):      # A + C
      if os.path.getsize(self.path + self.aFile) < self.maxlen:
        f = open(self.path + self.aFile, 'a+')
      else:
        os.remove(self.path + self.cFile)
        f = open(self.path + self.bFile, 'a+')
      f.write(tim + message +'\n')
      f.close()
    else:       # A + B + C
      pass
      os.remove(self.path + self.aFile)
      os.remove(self.path + self.bFile)
      os.remove(self.path + self.cFile)
```

Appendix B

Histograms of Delay Measurements

All histograms present notification delay measurements (RTT).

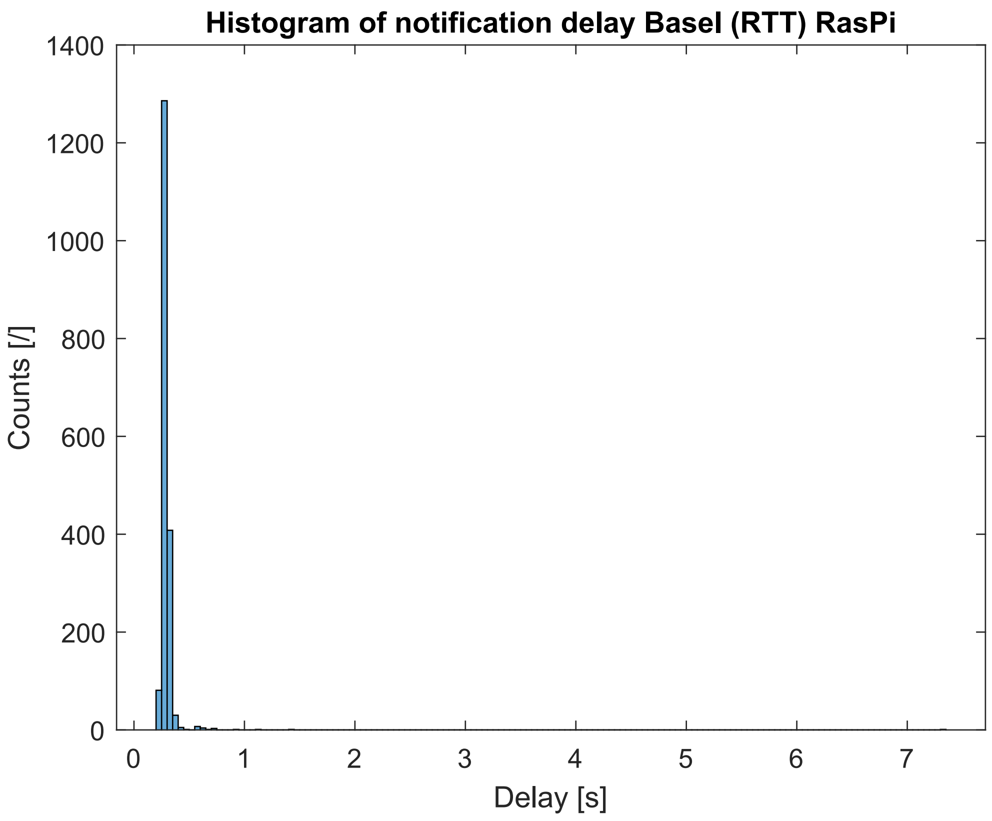

Figure B.1: Histogram of the notification delays from Basel, Switzerland, to Raspberry Pi server

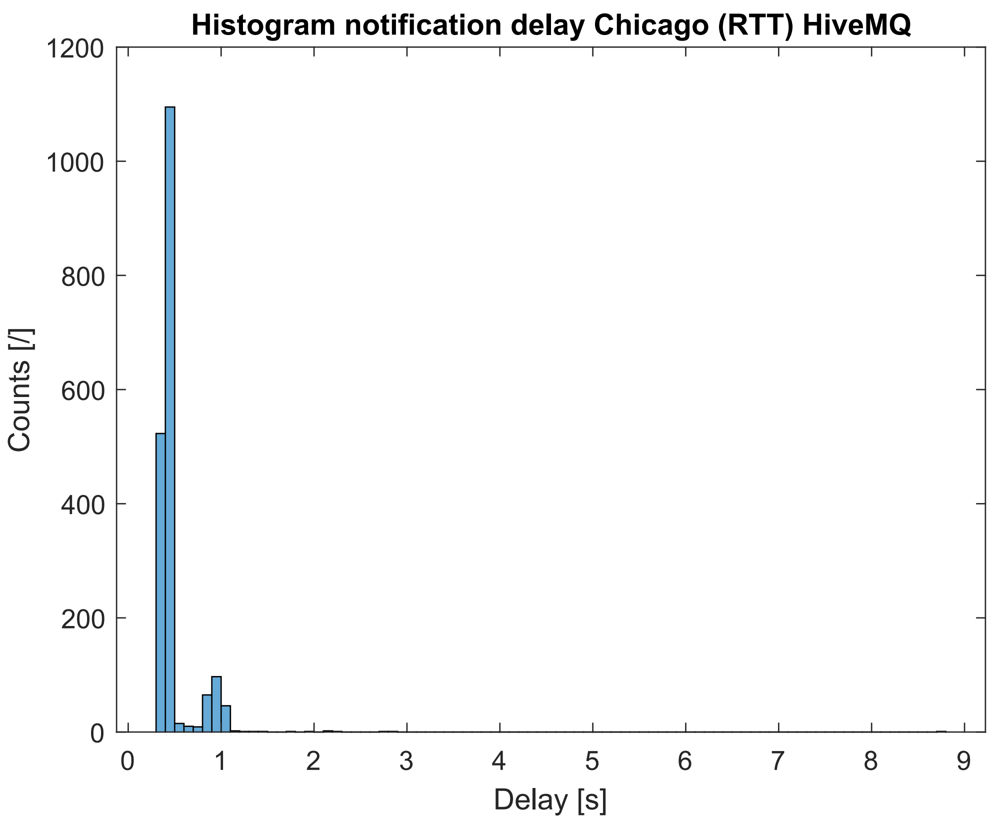

Figure B.2: Histogram of the notification delays from Chicago, USA, to HiveMQ server

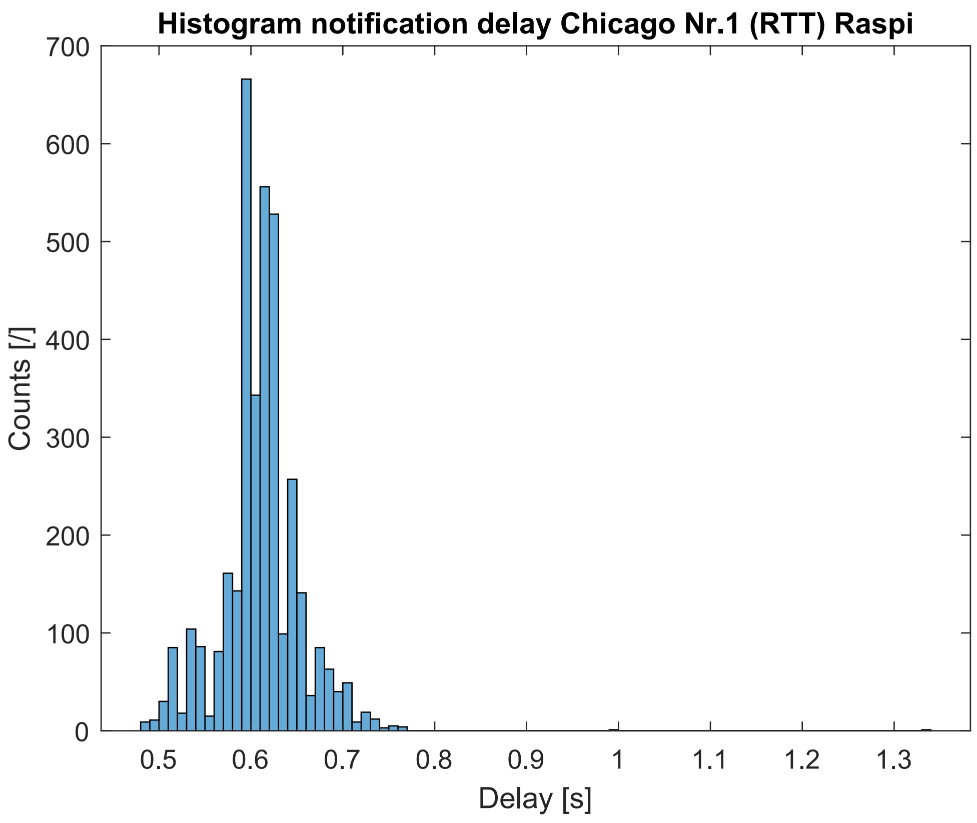

Figure B.3: Histogram of first measured notification delays from Chicago, USA, to Raspberry Pi server

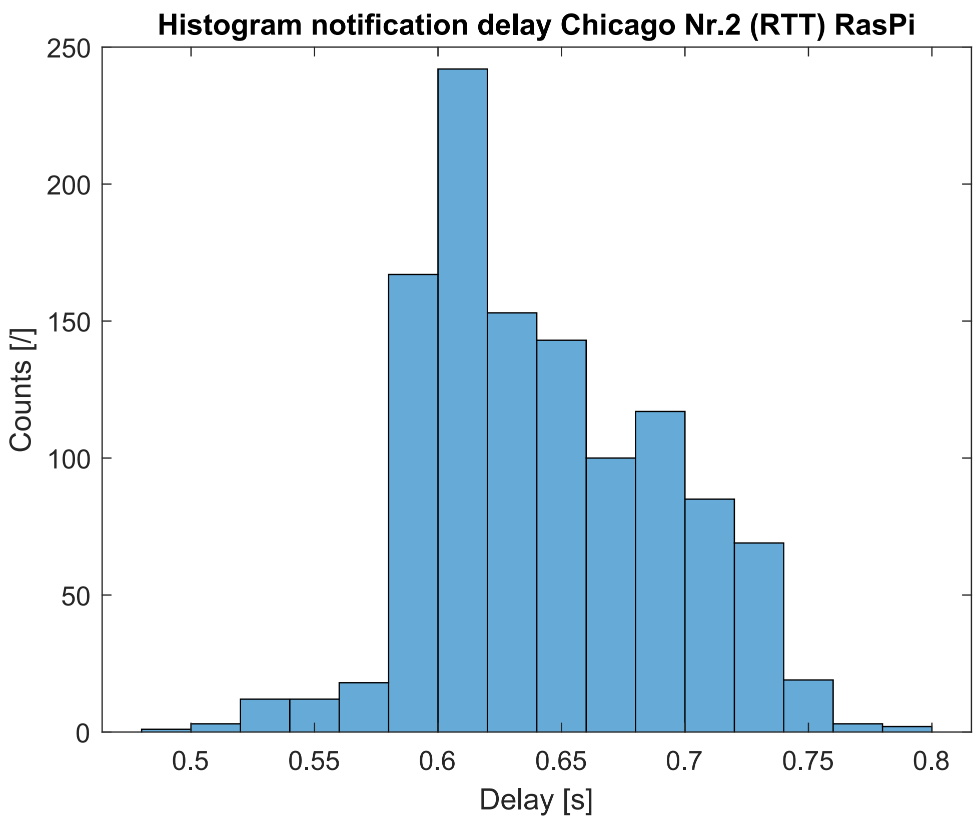

Figure B.4: Histogram of second measured notification delays from Chicago, USA, to Raspberry Pi server

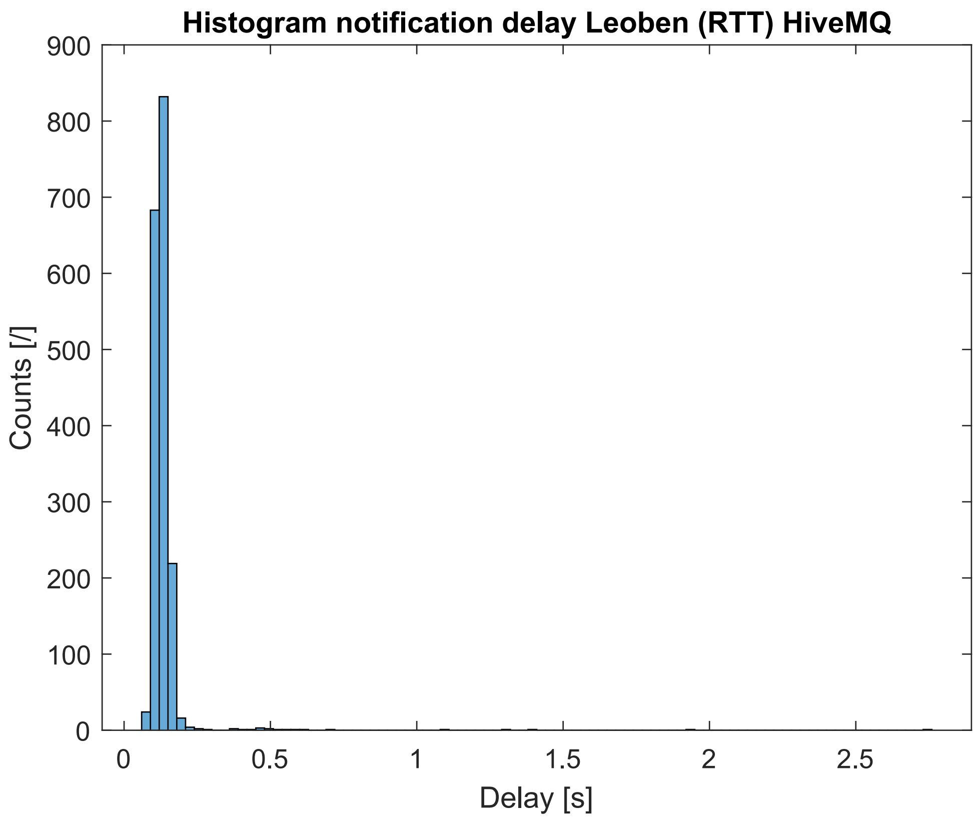

Figure B.5: Histogram of the notification delays from Leoben, Austria, to HiveMQ server

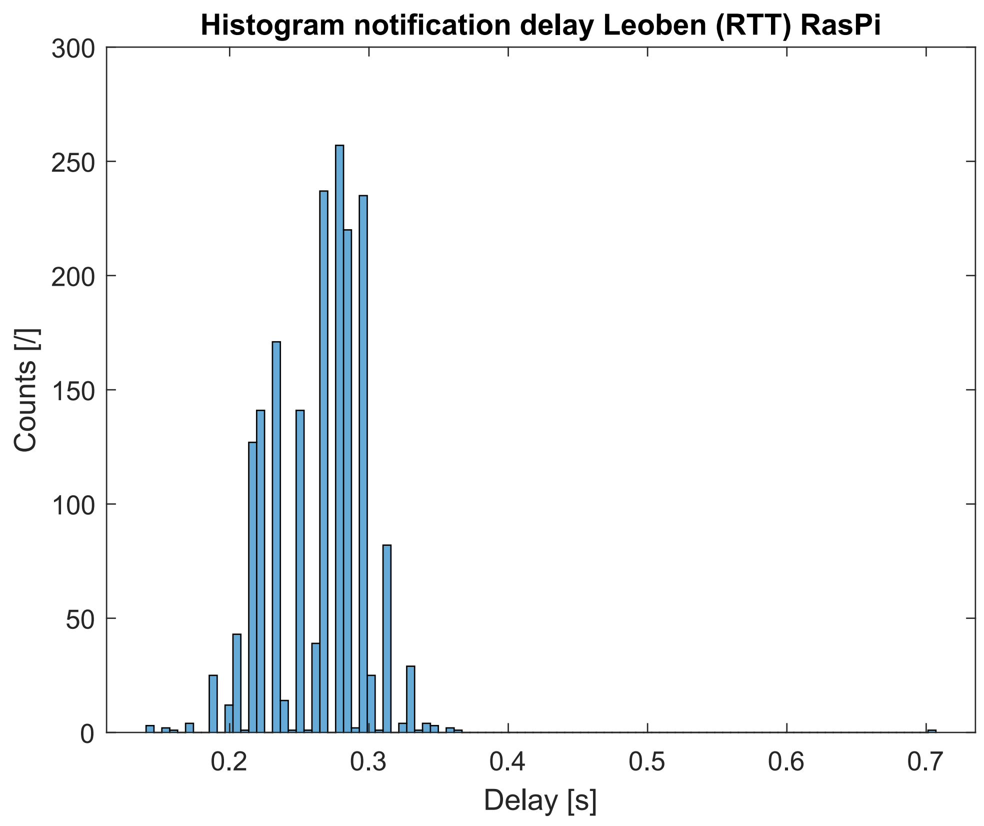

Figure B.6: Histogram of the notification delays from Leoben, Austria, to Raspberry Pi server

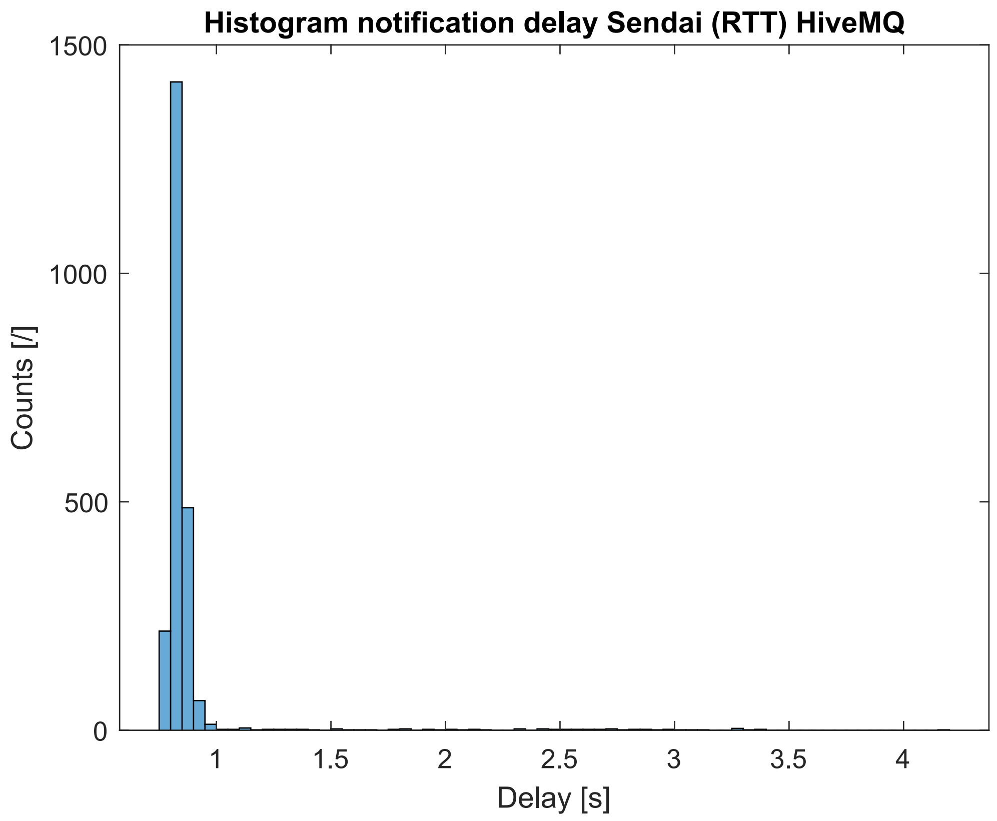

Figure B.7: Histogram of the notification delays from Sendai, Japan, to HiveMQ server

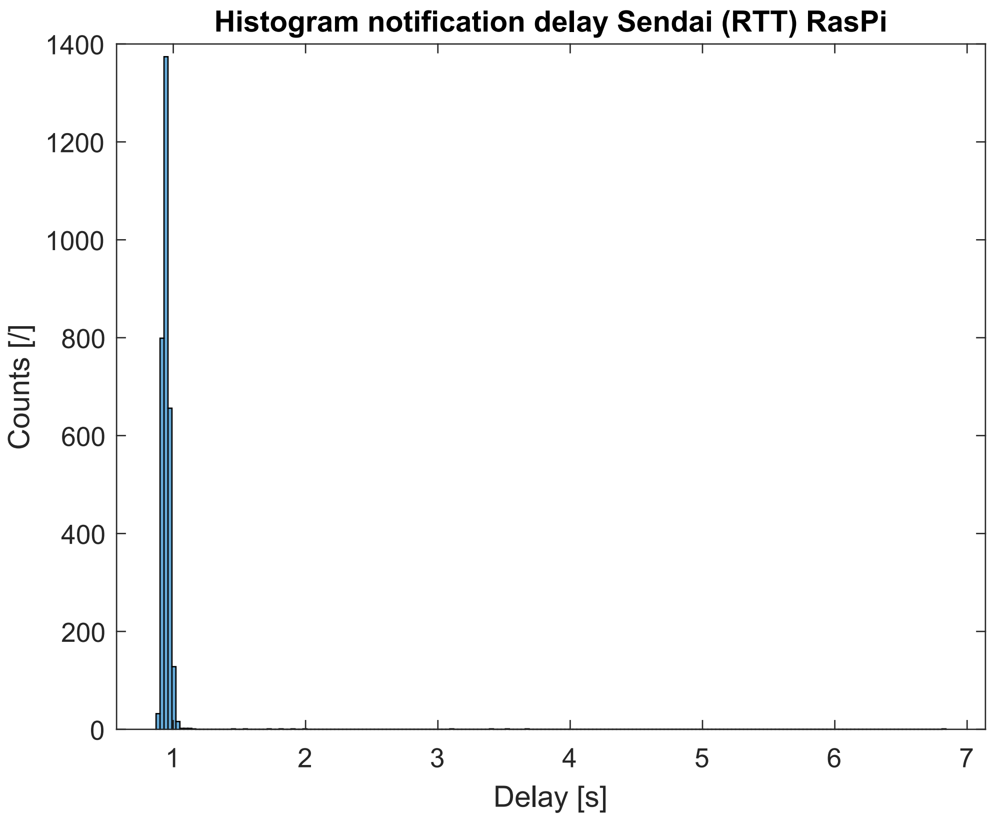

Figure B.8: Histogram of the notification delays from Sendai, Japan, to Raspberry Pi server

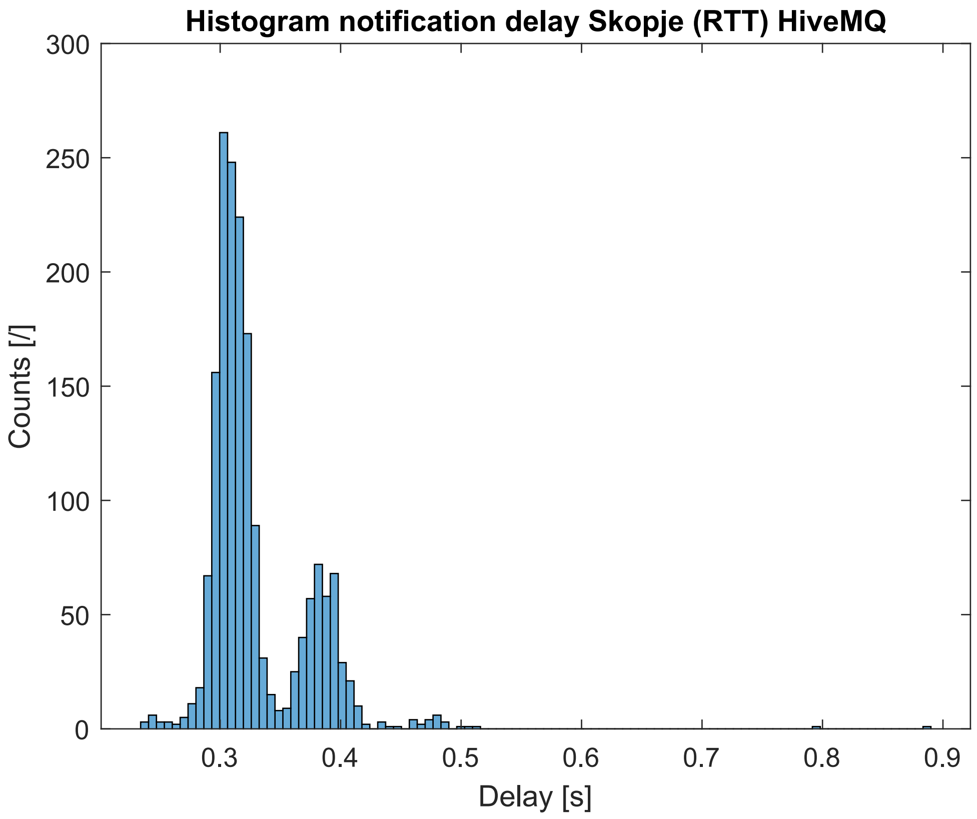

Figure B.9: Histogram of the notification delays from Skopje, North Macedonia, to HiveMQ server

Appendix C

Gateway

Listing C.1: Code for the Gateway to convert UDP into TCP

```
# Author : Gemith Mattathil
# Date : 15. February 2023
# Version : 3.0
#
# ESP8266
#
from time import sleep
from umqttsimple import MQTTClient
import ubinascii
import machine
import micropython
import network
import esp
esp.osdebug(None)
import gc
from machine import Pin, PWM, ADC
import utime as ut
import usocket as us
from struct import unpack, pack



gc.collect()

#-----------------------------------------
ssid = 'IoTmakesFun'
password = '************'
#-----------------------------------------
mqtt_server = 'broker.hivemq.com'
```

```
topic_sub   = b'mul/gemith/sysresponse'
topic_pub   = b'mul/gemith/controloutput'
topic_pubIP = b'mul/gemith/IP'
#----------------------------------------


client_id = ubinascii.hexlify(machine.unique_id())
station = network.WLAN(network.STA_IF)

station.active(True)
station.connect(ssid, password)

while station.isconnected() == False:
  pass

print('Connection successful')
print(station.ifconfig())
IPaddr = station.ifconfig()[0]

#--------------------------------------------------
output1 = 0; output2 = 0; output3 = 0;
gotmessage = False

def sub_cb(topic, msg):
  global output1, output2, output3
  global gotmessage
  print((topic, msg))
  if topic == topic_sub:
    try:
      outstr = msg.decode().split()
      output1 = int (outstr[0])
      output2 = int (outstr[1])
      output3 = int (outstr[2])
      gotmessage = True
    except:
      pass
    print('ESP received ', output1)

def connect_and_subscribe():
  global client_id, mqtt_server, topic_sub
  global IPaddr
  client = MQTTClient(client_id, mqtt_server)
  client.set_callback(sub_cb)
  client.connect()
  client.subscribe(topic_sub)
  client.publish(topic_pubIP, IPaddr   )
  print('Connected to %s MQTT broker, subscribed to %s topic' \
        % (mqtt_server, topic_sub))
  return client

def restart_and_reconnect():
  print('Failed to connect to MQTT broker. Reconnecting...')
  sleep(10)
  machine.reset()

try:
  client = connect_and_subscribe()
except OSError as e:
  restart_and_reconnect()

#------------------------------------------
receivedFromUdp = False
sendToUdp       = False
sendToMqtt      = False
udpPort        = 4841
udpAddr        = ''

sock = us.socket(us.AF_INET, us.SOCK_DGRAM)
sock.setblocking (False)
sock.bind((udpAddr, udpPort))




#---------------------------------------------------
starttime = ut.ticks_ms()


while True:
  #------------------  UDP ---------------------------
  try:
     data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
     print ("received message from: ", addr)
     controlsignal = int(unpack ('>d', data)[0])
     print ('From controller :', controlsignal)
     receivedFromUdp = True;
  except:
     pass
  if receivedFromUdp:
     receivedFromUdp = False
     sendToMqtt      = True
  if sendToUdp:
     udpmess  = pack ('<3d', output1, output2, output3)
     print ('To controller ', udpmess)
     sock.sendto (udpmess, (addr[0], 4841))
     sendToUdp = False

  #------------------- MQTT -------------------------
  try:
    client.check_msg()
    if gotmessage:
      gotmessage = False
      zeit = ut.ticks_ms()
      delta = zeit-starttime
      starttime = zeit
      print ('Zeit    ', delta)
      sendToUdp = True
  except OSError as e:
    restart_and_reconnect()
  if sendToMqtt:
      msg3 = str(controlsignal)
      client.publish(topic_pub, msg3)
      sendToMqtt = False
  sleep (0.01)
```

Appendix D

Gateway Influence

Listing D.1: Code for recording the measurement time difference with and without Gateway

```
# Author : Gemith Mattathil
# Date : 16. February 2023
# Version : 2.0

import paho.mqtt.client as mqtt
import paho.mqtt.publish as publish
import socket
import struct
from time import sleep
import time
import sys
from logger import Logger
from re import sub

if len (sys.argv) != 3:
  print ("""Needs argument: M or U (MQTT or UDP)
                            name of logfile!
          """)
  sys.exit ()
logfilename = sys.argv[2]
method      = sys.argv[1]



#================  experiment parameters ================
numberOfMeasurements = 100 #3600
pauseTime            = 2.0
commentLine  = 'MQTT response test: delay to end of subscription, \
  delay to end of notification'
```

```
commentLine2 = 'UDP response test: delay to end of subscription, \
  delay to end of notification'

out = str(101)

#=============================  udp =========================
UDP_IP2 = '192.168.1.147'
UDP_PORT2 = 4841 #25000
MY_IP    = ''

MY_PORT   = 4841 # 25001
message = 10.0
sock = socket.socket (socket.AF_INET, socket.SOCK_DGRAM)
#adressfamilie #internet, udp datagrams
sock.bind (('', MY_PORT)) # simulink will es so
sock.setblocking (False)

#==================  subscriber =============
MQTT_SERVER   = "broker.hivemq.com"     #broker!
MQTT_SUBTOPIC = "mul/gemith/IP"    # topic!
MQTT_PUBTOPIC_CONTROL  = "mul/gemith/controloutput"
MQTT_SUBTOPIC_RESPONSE = "mul/gemith/sysresponse"

dataMQ = []



start = 0.
notification_time = 0.
started = False
state = 'MQ'   #'UDP'
receivedFromMQ  = False
receivedFromUdp = False

def messMQ (out):
  publish.single(MQTT_PUBTOPIC_CONTROL, out, qos =0, \
               hostname=MQTT_SERVER)
def messUDP (out):
  global sock
  ba = bytearray(struct.pack(">d", float(out)))
  print (out, ba)


  sock.sendto (ba, (UDP_IP2, UDP_PORT2))

# The callback for when the client receives a
# CONNACK response from the server.
```

```
def my_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe(MQTT_SUBTOPIC_RESPONSE, qos=0)

# The callback for when a PUBLISH message is received from the server.
def my_message(client, userdata, msg):
    global dataMQ
    global receivedFromMQ
    receivedFromMQ = True
    #global state_init
    result = str(msg.payload.decode("utf-8"))
    dataMQ= result
    print(msg.topic+" ", result)
    # more callbacks, etc

client = mqtt.Client()
client.on_connect = my_connect
client.on_message = my_message

client.connect(MQTT_SERVER, port=1883, keepalive=60)
client.loop_start ()



log = Logger (logfilename)
log.setMaxlen(1000000)
if method == 'M':
  log.log (commentLine)
elif method == 'U':
  log.log (commentLine2)
sleep(1)

numMeas = 0

while True:
  started = True
  now = time.time ()
  zeit = now - start
  if zeit >= pauseTime:
    start = now
    if method == 'M':
      messMQ (out)
    elif method == 'U':
      messUDP (out)
    numMeas += 1

  try:
    data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
    print ("received message from: ", addr)
    dataUDP = unpack ('>ddd', data)
    sysrespOut  = int(dataUDP)[0]
    sysrespIn   = int(dataUDP)[1]
    sysrespTime = int(dataUDP)[2]
    print ('From controller :', dataUDP)
    receivedFromUdp = True;
  except:
     pass
  if receivedFromUdp:
    receivedFromUdp = False
    print ('From controller :', dataUDP)
    log.log (str(delay)+', '+dataUDP)
  if receivedFromMQ:
    receivedFromMQ = False
    delay = now - start
    data2 = dataMQ.replace (' ', ', ')
    log.log (str(delay)+', '+data2)
  #sleep (pauseTime)

  if numMeas >= numberOfMeasurements:
    break

sleep (2)  #wait for pending MQTT data
```

Appendix E

Circuit System

Listing E.1: Code for second order system functioning as a plant in the control system

```
# Author : Gemith Mattathil
# Date : 06. March 2023
# Version : 2.0
# ESP32
#
from time import sleep
from umqttsimple import MQTTClient
import ubinascii
import machine
import micropython
import network
import esp
esp.osdebug(None)
import gc
from machine import Pin, PWM, ADC
import utime as ut

adc = ADC(Pin(32))
pwm = PWM (Pin (33))
pwm.freq (10000)
duty = 200
pwm.duty (duty)

gc.collect()

ssid = 'IoTmakesFun'
password = '*************'

mqtt_server = 'broker.hivemq.com'

```

```
client_id = ubinascii.hexlify(machine.unique_id())
topic_sub = b'mul/gemith/controloutput'
topic_pub = b'mul/gemith/sysresponse'

last_message = 0
message_interval = 5
counter = 0

station = network.WLAN(network.STA_IF)

station.active(True)
station.connect(ssid, password)

while station.isconnected() == False:
  pass

print('Connection successful')
print(station.ifconfig())

output = 0
gotmessage = False

def sub_cb(topic, msg):
  global output
  global gotmessage
  print((topic, msg))
  if topic == topic_sub:
    try:
      output = int (msg.decode())
      gotmessage = True
    except:
      output = 0
    print('ESP received ', output)

def connect_and_subscribe():
  global client_id, mqtt_server, topic_sub
  client = MQTTClient(client_id, mqtt_server)
  client.set_callback(sub_cb)
  client.connect()
  client.subscribe(topic_sub)
  print('Connected to %s MQTT broker, subscribed to %s topic' \
        % (mqtt_server, topic_sub))
  return client

def restart_and_reconnect():
  print('Failed to connect to MQTT broker. Reconnecting...')
  sleep(10)
  machine.reset()

try:
  client = connect_and_subscribe()
except OSError as e:
  restart_and_reconnect()

def limit (value):
    if value > 511:
        out = 511
    elif value < 0:
        out = 0
    else:
        out = int(value)
    return (out)

starttime = ut.ticks_ms()
lastpingtime = starttime
#======================================
while True:

  try:
    client.check_msg()
    if gotmessage:
      gotmessage = False
      duty = limit(output)
      pwm.duty (duty)
      zeit = ut.ticks_ms()
      delta = zeit-starttime
      starttime = zeit
      print ('Zeit    ', delta)
      sleep (0.01)
      msg = str(adc.read()) + ' ' + str(output) + \
        ' ' + str(ut.ticks_ms())
      client.publish(topic_pub, msg)
      lastpingtime = zeit
  except OSError as e:
    restart_and_reconnect()

  pingtime = ut.ticks_ms()
  if (pingtime - lastpingtime) > 3000000: # ms
    try:
        client.ping()
    except:
        restart_and_reconnect()
    print ('Ping!')
    lastpingtime = pingtime
```

Appendix F

Matlab Code for the Simulation

Listing F.1: Matlab code for the simulation of the system with staochastic delay

```matlab
% Author : Gemith Mattathil
% Date : 18. March 2023
% Version : 3.0


%% Setting up workspace
clear
close all
clc

%% System parameters
C1 = 1500e-6;
C2 = 1000e-6;
Unom = 12;
R1 = 1000;
R2 = 1000;

A = [-1/(C1*R1)-1/(C1*R2)   1/(C1*R2) ; ...
    1/(C2*R2)                -1/(C2*R2)];

B = [ 1/(C1*R1)  0]';
C = [0 1];  % look at output only

%% observer feedback matrix
```

```
p1= [-0.7 -0.5]; % poles of the observer
L = (place (A', C', p1))';


%% Initialisations
Tnom = 1;
tstart = 0;
tend = 50;
Tlocal = 0.01;

setval = 1000;
kp = 0.1;
Tr = 10;

x0 = [0; 0];
i0 = 0;
x01 = x0;
x0d = x0;
u0d = 0;
yd = 0;
mess_y = 0;
mess_u = 0;
stime = 0;
stime_old = 0;

tplot = 0; uplot = 0; yplot = 0; tsample = 0;
n = 0;
nsample = 1;
comcycle = 1;

%% delay planning:
delConSys = rand (1, tend/Tlocal)*0.4 + 0.4;
delSysCon = rand (1, tend/Tlocal)*0.4 + 0.4;

C_ACTIVE = true;    % controller should send at instant of
   flag
C_SENDTRAVEL = false;
C_WAITING  = false;
```

```
S_ACTIVE  = false;  % system has received
S_WAITING = false;
S_SENDTRAVEL = false;

c_done = false;
s_done = false;
creadytime = -0.1;


for time = tstart:Tlocal:tend
    % communication state machine
    if C_ACTIVE
        mess_y = yd;
        if c_done
            C_ACTIVE = false; C_SENDTRAVEL = true;
            creadytime = time; c_done = false;
        end
    end
    if C_SENDTRAVEL
        if time > creadytime + delConSys (comcycle)
            C_SENDTRAVEL = false;
            S_ACTIVE = true;
            s_done = false;
            sacttime = time;
        end
    end
    if S_ACTIVE
        mess_u = u;
        if s_done
            S_ACTIVE = false; S_SENDTRAVEL = true;
            sreadytime = time; s_done = false;
        end
    end
    if S_SENDTRAVEL
        if time > sreadytime + delSysCon (comcycle)
            S_SENDTRAVEL = false;
            C_ACTIVE = true;
            comcycle = comcycle + 1;
        end
    end

    % activities of system and controller
    if C_ACTIVE
        if time > creadytime + Tnom
            disp ('contr active'); time
            c_done = true;
            t0 = creadytime; t1 = time;
            [u, i0] = picontr2 (kp, Tr, i0, setval, mess_y,
                t0, t1);
        end
    end
    if S_ACTIVE
           disp ('sys active'); time
           s_done = true;
           u0d = mess_u;
           t0 = stime_old; t1 = time;
           [yd, x0d] = process(A, B, C, x0d, u0d, t0, t1);
           stime_old = time;
           stime = time;
           %mess_y = yd;
           mess_time = stime;
           tsample (nsample) = time;
           nsample = nsample+1;
           textmes = sprintf ("t: %.2f u: %.1f y: %.1f",
              time, u0d, yd)
    end

    n = n+1;
    tplot (n) = time;
    yplot (n) = yd;
    uplot (n) = u0d;

end

%% Plotting
 figure
 plot (tplot, uplot)
 grid on
 title ('Controller output over system time')
 xlabel ('Time [s]')
 ylabel ('Magnitude')
 figure
 plot (tplot, yplot)
 title ('System response over system time')
 xlabel ('Time [s]')
 ylabel ('Magnitude')
 grid on

%% Functions
function [y, x] = process(A, B, C, x0, u, t0, t1)

if u < 0
    u = 0;
elseif u > 511
   u = 511;
end
u = u*10.5; %3.3*4096/1024;  %scale to adc and dac
dim   = length (B);
expo0 = expm (A*(t1-t0));
conv0 = (expo0 - eye(dim))*inv(A) * B;
x = expo0*x0 + conv0*u;
y = C*x;
if y < 0
    y = 0;
elseif y > 4095
    y = 4095;
end
end


function [u, i0] = picontr2 (kp, Tr, i0, setval, actval, t0
   , t1)
e = setval - actval;
u = kp*(e+i0);
i0 = i0 + e*(t1-t0)/Tr;
end
```