

An Evaluation of IoT Application Protocols

Kristian Fatohi

Main field of study: Computer Science

Credits: 15 hp

Semester/year: Spring: 2023

Supervisor: Stefan Forsström

Examiner: Patrik Österberg

Course code: DT099G

At Mid Sweden University, it is possible to publish the thesis in full text in DiVA (see appendix for publishing conditions). The publication is open access, which means that the work will be freely available to read and download online. This increases the dissemination and visibility of the degree project.

Open access is becoming the norm for disseminating scientific information online. Mid Sweden University recommends both researchers and students to publish their work open access.

I/we allow publishing in full text (free available online, open access):

- ☒ Yes, I/we agree to the terms of publication.
- ☐ No, I/we do not accept that my independent work is published in the public interface in DiVA (only archiving in DiVA).

Sundsvall: 2023-06-09

Location and date

DT099G

Programme/Course

Kristian Fatohi

Name (all authors names)

2001-05-06

Year of birth (all authors year of birth)

Abstract

The Internet of Things is a concept that has gained widespread attention and adaptation all around the world, but few people really do understand how it works and what it really is. The concept revolutionized the way we interact with our surroundings, it revolves around connectivity between smart devices and how they communicate with other devices or with humans. To ensure effective communication, a set of application layered protocols has been developed. These protocols are designed to be beneficial in certain areas, which is why the objective for this study has been to evaluate and examine the performance of existing application communication protocols. A thorough literature review is conducted to identify and gain a deeper understanding of their unique characteristics and features for some protocols that exist and are used today. All protocols brought up in this thesis will not be examined and choosing which protocols to inspect further was done by doing a comparative analysis where factors such as, communication method and payload limit size were taken into consideration. Performance assessment was done for the protocols that qualified, where factors like latency, throughput, and scalability were measured. The results of these tests are used to draw conclusions about the suitability of each protocol. After a comprehensive evaluation based on experiments with simulations and literature reviews, this thesis concludes that MQTT and CoAP as the most suitable protocols for general IoT applications due to their lightweight, efficient, and scalable nature.

Keywords: Quality of Service (QoS), IoT (Internet of Things).

Sammanfattning

Hur vi människor kommer att leva i framtiden kan kraftigt påverkas utav konceptet Internet of Things. IoT erbjuder unika möjligheter för att hjälpa oss människor att automatisera och förenkla vissa ärenden. Även om konceptet har funnits ett tag så har man fortfarande vilseledda åsikter kring potentialen med IoT. Det intressanta med konceptet och vad som enligt mig är den fundamentala grunden för IoT är självaste kommunikationen. Men hur går kommunikation till? Ett flertal IoT kommunikationsprotokoll har utvecklats för att försöka förbättra kommunikationen mellan två och flera smarta enheter. Eftersom det finns ett par olika kommunikationsprotokoll så är målet för denna studie att utvärdera och undersöka prestandan för befintliga applikations lagrade kommunikationsprotokoll. En litteraturrecension görs för att identifiera och få en djupare förståelse för deras unika egenskaper som skiljer protokollen åt. Alla protokoll som tas upp i denna rapport kommer inte att undersökas och valet av vilka protokoll som ska inspekteras närmare gjordes genom en jämförande analys där faktorer som kommunikationsmetod och begränsningen av nyttolast storleken beaktades. Latens, genomströmning och skalbarheten är de prestandafaktorerna som bedöms för de protokoll som valts för en närmare inspektion. Resultaten av dessa tester används för att dra slutsatser om lämpligheten för varje protokoll. Efter en omfattande utvärdering baserat på de experimenten som gjorts under projektets gång så drar denna rapport slutsatsen att MQTT och CoAP är de mest lämpliga protokollen för allmänna IoT-applikationer på grund av deras lätta, effektiva och skalbara natur.

Nyckelord: Quality of Service (QoS), IoT (Internet of Things).

Acknowledgements / Foreword

I would like to express my sincere gratitude to my supervisor, Stefan Forsström, for his valuable guidance, support, and mentorship throughout the completion of my bachelor's thesis.

Table of Contents

1	Introduction	1
1.1	Background and motivation	1
1.2	Overall aim and problem statement	1
1.3	Scientific goals/Research questions	2
1.4	Scope	2
1.5	Outline	2
2	Theory	4
2.1	Internet of Things	4
2.2	IoT Devices	4
2.2.1	Raspberry Pi	5
2.3	Performance Factors	5
2.3.1	Latency	5
2.3.2	Throughput	5
2.3.3	Scalability	6
2.4	Related work	6
2.4.1	Comparative Study of IoT Protocols	6
2.4.2	Comparison of IoT Applications Layer Protocols	6
3	Methodology	8
3.1	Scientific method description	8
3.2	Project/Work method description	9
3.3	Project evaluation method	10
4	Pre-study	11
4.1	Solution alternatives	11
4.1.1	Hypertext Transfer Protocol (1991)	11
4.1.2	Message Queue Telemetry Transport (1999)	13
4.1.3	Extensible Message and Presence Protocol (1999)	15
4.1.4	Constrained Application Protocol (2010)	17
4.1.5	Matter	19
4.2	Comparison of solutions	21
4.3	Chosen solution	21
5	Implementation	23
5.1	Message Queue Telemetry Transport	23
5.1.1	MQTT Publish Client	24
5.1.2	MQTT Subscribe Client	27
5.1.3	MQTT Broker	29
5.2	Constrained Application Protocol	29
5.2.1	CoAP Client	30

5.2.2 CoAP Server	31
5.3 Matter.....	32
5.4 Evaluation setup	33
6 Results	34
6.1 Resulting system	34
6.2 Measurement results	35
6.2.1 Latency	35
6.2.2 Throughput	36
6.2.3 Scalability.....	38
7 Discussion.....	40
7.1 Analysis and discussion of results	40
7.2 Project method discussion	42
7.3 Scientific discussion.....	42
7.4 Ethical and societal discussion.....	43
8 Conclusions	44
8.1 Future Work.....	44
8.1.1 Deep Dive into the Matter Protocol	44
8.1.2 Further Performance Comparison Between CoAP and MQTT	45

Terminology

Abbreviations

6LoWPAN IPv6 over Low-power Wireless Personal Area Network

ACK Acknowledge

API Application Programming Interface

CHIP Connected Home over IP

CoAP Constrained Application Protocol

DSRM Design Science Research Methodology

IBM International Business Machine Corporation

IETF Internet Engineering Task Force

IoT Internet of Things

IP Internet Protocol

IPv6 Internet Protocol version 6

M2M Machine-to-Machine

MQTT Message Queue Telemetry Transport

SSE Server-Sent Events

TCP Transmission Control Protocol

TCP/IP Transmission Control Protocol/Internet Protocol

UDP User Datagram Protocol

WWW World Wide Web

XML Extensible Markup Language

1 Introduction

If history has ever taught us humans anything, then it must be that changes occur very often. The world we live in today was certainly very different to how it was years ago and will surely be immensely different years ahead. Occasionally, a new phenomenon appears, technology is one of those phenomena. Technology is like the sea, there is so much of it yet, so little is discovered. The possibilities with technology are endless and every day we take a step towards a newer concept.

1.1 Background and motivation

The Internet of Things (IoT) is one of those concepts that could potentially change the way we humans live. IoT can connect two physical devices over the internet so that they can communicate with each other. Imagine having your desk lamp communicating with your mobile device, or maybe the vacuum cleaner in your home being able to listen to a tablet of yours. The potential with IoT is clearly visible but it also presents some potential dangers and risks. IoT devices often have security threats and privacy concerns and due to this there is a limit to the limitless potential IoT has.

The most well-known smart home devices such as Apple HomeKit, Google Home and Amazon Alexa are cloud-based systems, meaning that they run on a cloud computing platform, accessed via the internet, rather than on a local computer or server. This system provides scalable, flexible, and cost-effective computing resources but what if the cloud goes offline? What happens then to the cloud-based devices? Users may experience complete data loss and the device may not be accessible by the user anymore. To ensure that this will not happen, data can be stored locally.

1.2 Overall aim and problem statement

There are several communication protocols for IoT devices, two of the most used are Constrained Application Protocol (CoAP) and Message Queuing Telemetry Transport (MQTT). The matter protocol, formerly project Connected Home over IP (CHIP) on the other hand is the newest communication protocol for IoT devices and has not really been used at full capacity yet. Further investigation of the newly and potentially dominated communication protocol is needed and therefore is this project's overall aim to examine if the newer communication protocols are better in terms of performance compared to the older and more

standard communication protocols so that the newer protocols can be implemented and change the user's experience for the better.

The investigated problem statement of this thesis is that newer communication IoT protocols have not really been analyzed in a wide scale, and it may take some time for academic institutions and researchers to publish their finding and analyses on these protocols. Also, because the protocols are new, they may have not gained widespread recognition and attention as a topic for academic studies. Therefore, I will dive into the new protocols to see how different they are too other more used protocols, in terms of performance and implementation to see if the newer protocols can be more beneficial in certain areas.

1.3 Scientific goals/Research questions

This study will achieve a better understanding of the answers to the following questions:

1. Which are the most prominent IoT communication protocols and which areas can benefit the most from using newer IoT protocols?
2. What performance can these protocols get in terms of throughput, latency, and scalability?
3. How limited is the documentation for developing an open-source client for the newer communication protocols, in comparison to the older and more established protocols?

1.4 Scope

This research will focus on one of the newer communication protocols and analyze its benefits and drawbacks, if any. Therefore, older communication protocols will not be the focus. When evaluating the performance of the communication protocols, all the factors that can be evaluated will not get any attention, such as the security aspect as well as interoperability.

1.5 Outline

Chapter two describes relevant theory that makes this thesis easier to understand, where concepts like Internet of Things and Web of Things are explained meticulously. Chapter three goes over how the research questions will be answered in the scientific and project method. Chapter four discusses the pre-study done to learn about existing protocols.

Chapter five describes how the implementation was build and. Chapter six presents the results of the implementation. Chapter seven discusses the results, the choice of method, ethical and scientific knowledge. Finally, chapter eight concludes the thesis with a conclusion and suggestion for future work.

2 Theory

All the theoretical and background material required to communicate a clearer grasp of the problem motivation will be covered in this chapter. First, the Internet of Things and its importance will be covered then this chapter will cover what IoT device that will be used throughout the project and then how the performance factors that are in focus will be measured, which will provide the reader background knowledge necessary to understand the performance test. Lastly, similar studies will be covered to understand what resemblance there are with this study and what differs.

2.1 Internet of Things

The Internet of Things or IoT is a revolutionized communication concept. IoT refers to a system of a global scale with interconnected computer networks. These networks communicate with each other by utilizing the standardized rules that are formed by the TCP/IP model. IoT offers connectivity for almost any object whenever needed [1]. The networks consist of physical devices such as vehicles, home appliances, and other items embedded with sensors and software which enables these objects to collect and exchange data [2]. This data exchange can occur either by human-to-human, human-to-object, or object-to-object communication [1]. A person required to have a heart monitor implant is an example of a human-to-human interaction. The information that devices share with one another may be state information about another device or just collected data from sensors scattered in an environment. The exchanged data will then be stored and analyzed so it later can be used by applications or other devices. [3]

In summary, as mentioned earlier, Internet of Things is a concept based of the TCP/IP communication model where the focus lays on the network layer. This is because IoT's main concern is to enable efficient and seamless communication between physical devices. This required communication can only be achieved by having a strong connectivity between the devices. This connectivity is then provided by the IoT.

2.2 IoT Devices

Internet of Things devices are physical devices that are connected to the internet and can achieve a connection with other connected devices. This study uses the aid of an IoT device namely, a Raspberry Pi.

2.2.1 Raspberry Pi

A Raspberry Pi is a small, inexpensive computer that connects to either another computer or a TV. The Raspberry Pi operates using a regular keyboard and mouse. The United Kingdom foundation, Raspberry Pi Foundation, made the small computer, Raspberry Pi. This device has all the features of a standard desktop computer which includes all from browsing on the internet to playing games. [4]

2.3 Performance Factors

There are several performance factors when measuring a communication protocol. Those that are in focus in this thesis are explained further.

2.3.1 Latency

The amount of time it takes for data to move between two points on a network is known as latency. Suppose Server A and Server B and A want to send a data packet to B. At 05:24:00:000 GMT, Server A delivers the data packet and at, 05:24:00:100 GMT, Server B receives the data packet. The difference between the two times is the amount of latency, in this case it is 0.100 seconds or 100 milliseconds [5]. This is visualized by figure 1 below.

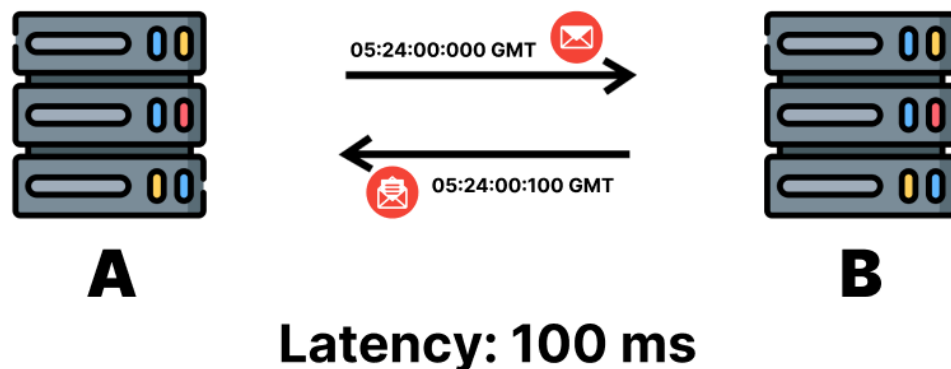


Figure 1: An example of a latency measurement

2.3.2 Throughput

Throughput describes the amount of data that has been successfully transported from a source to a destination in a specific amount of time and is therefore often expressed in terms of bits per second (bps) or a multiple of bps, such as kilobits per second (Kbps) or megabits per second (Mbps) [6]. Suppose a network has a throughput of 10 Mbps and

a transfer of a video file is about to take place, the file is 100 megabytes in size. This transfer will then take around 80 seconds, excluding competing traffic, to complete in a 10 Mbps network. However, the throughput measurements in this project will be measured through messages sent per second, this is also called system throughput [7].

2.3.3 Scalability

A protocol is said to be scalable if it can manage growing volumes of data traffic without experiencing a performance decrease. For instance, a protocol to manage a high volume of data from numerous sensors and devices may be required in the context of a smart city. A protocol that is not scalable can have trouble keeping up with the increased traffic, causing delays or missed packets that might cause data loss and worse system performance.

2.4 Related work

In order to provide both a larger and deeper understanding of the issue, this subchapter will review relevant works that address the same or comparable challenges.

2.4.1 Comparative Study of IoT Protocols

Sakina Elhadi, Abdelaziz Marzak, Nawal Sael, and Soukaina Merzouk wrote the article “Comparative Study of IoT Protocols” [8]. This study examines some of the most significant IoT protocols. It concentrates especially on layer network protocols and application protocols. Their study evaluates the protocols capabilities and contrasts their key traits and action in terms of various criteria's. This was done by doing a comparative study of all the protocols included to then do a comparative table to demonstrate the differences and similarities between each protocol and criteria. [8]

Their article is related to this thesis in terms of comparing the IoT protocols, but their comparison is only done by a comparative study while this study also compares a selected few by doing tests to calculate the latency, throughput, and scalability. Their article does not include some newer IoT protocols due to it being published in 2018 and the biggest difference between our projects is that an open-source client will be created for a newer protocol.

2.4.2 Comparison of IoT Applications Layer Protocols

Pinchen Cui submitted her master thesis “Comparison of IoT Application Layer Protocols” to the graduate faculty of Auburn University [9]. The motivation for Pinchen Cui’s thesis is to offer a fundamental framework for selecting an IoT application layer protocol by providing sufficient results from performance tests and evaluating them. MQTT, CoAP, DDS, and AMQP are the four application layer protocols that are being compared. This comparison is done at concept level meaning that the comparison is being made at a high level of abstraction rather than focusing on specific details or implementations. This is done by setting up five raspberry Pis, one for a common gateway device and a raspberry pi each for the protocols. The common gateway device communicates to the web server while the raspberry Pis for the protocols connects to the common gateway device via Wi-Fi. [9]

The method Cui used to compare the protocols is quite similar to the one this thesis will use to compare the chosen protocols. Due to Cui’s thesis being published in 2017, newer protocols that exist today were not compared which will instead be compared in this thesis. An open-source client will also be built on a newer protocol in this thesis which was not a part of Cui’s thesis.

3 Methodology

The following chapter describes the proposed methodology and research design of the thesis to properly answer the questions in chapter 1.

3.1 Scientific method description

The scientific method this study will use follows a quantitative method for the research. The steps for the method will be inspired by some of the steps included in the design science research methodology (DSRM). DSRM focuses on producing and analyzing answers to contemporary issues by way of the production of innovative artifacts. The design science research methodology involves iterative processes such as problem identification, design, implementation, and evaluation.

The first research question was related to which IoT communication protocols exist. The main aim for this objective is to gather insight into what a communication protocol is and its purpose and in addition, several communication protocols may have different intentions for different tasks. After gaining the knowledge from the research to answer that question, the follow-up of the first research question can be answered, specifically which areas can benefit the most from using newer IoT protocols. The solution to these questions will be approached by doing a quantitative study with the first step being a literature review. After the quantitative study an observational study will take place to examine the areas where newer IoT protocols can benefit the most from.

Next research question is associated with to find out what performance these protocols can get in terms of throughput, latency, and scalability. This goal will mainly focus on a limited amount of IoT communication protocols, and preferably will the chosen protocols be different from each other regarding the way the protocols are developed. The limited number of protocols that will be chosen will be heavily influenced by the observation study from the previous research question. I will attack this question by conducting experiments for each performance factor and its respective protocol. With the results from the experiments an evaluation will occur where the outcome from the tests for each protocol will be compared to each other by a mathematical analysis.

The last research question corresponds to compare how an open-source client for newer protocols can be developed. This will be like the previous

research question, only focusing on a limited number of protocols, namely one newer protocol. To make the comparison and start the development of the open-source client, research in the form of a literature review will take place to find if there is any documentation on how to set up an open-source client for the protocol since it is new. With enough information regarding the development of the protocol, the prototyping part will begin. This step will include creating a proof-of-concept prototype of an open-source client for the protocol. Lastly, evaluating the process as a whole and comparing it to other more adapted protocols.

3.2 Project/Work method description

The methodology employed in this study will involve several steps. The initial step is to undertake a problem formulation exercise, which is crucial for defining and expressing the problem that requires a solution. This step is also important for identifying and outlining the requirements that will guide the subsequent steps of the study. Having a clear understanding of the problem and the requirements will enable me to develop an effective solution that addresses the identified challenges comprehensively. For example, this study's first research question will begin being answered by doing sort of a problem formulation more specifically, preparation research of the existing protocols to gain the understanding needed to answer the question.

After getting a better understanding of the problem and the possible solution, the implementation of the solution is next whether it being a testable product or answering a question. Research question two of this study depends on testing three protocols, this question will have its main focus on this milestone because it relies on the testable implementations of the chosen protocols. With a successfully completed implementation, the testing phase can begin. Then the results of the tests will be observed and evaluated, which is the next milestone in this study's methodology.

The evaluation phase includes examining the implemented solution whether it being by the factor's latency, throughput, and scalability as in research question two or how well and informative an implementation of an open-source client can be regarding other implementations of more well known and documented protocols as in research question three. In general, this phase is based around gathering data from tests and experiments so an analysis can be performed which will produce the

necessary results to make a sufficient enough statement on the question being answered.

3.3 Project evaluation method

The evaluation will be based on whether the three measurements taken of the protocols can be sufficient enough to determine if the thesis was a success. The quantitative values gathered from the tests would also provide valuable feedback so answers to the research questions can be produced.

4 Pre-study

The overall aim for this study is to improve the user's experience with IoT communication protocols for the better in terms of effectiveness and efficiency. Currently, many widely adopted communication protocols are outdated and infrequently updated. Conversely, newer protocols that show potential have yet to gain widespread attention. These protocols may, with sufficient enough tests, be a better option in the future for IoT communication devices. To gain a comprehensive understanding of the available protocols, a comparative study will be conducted as a pre-study to answer the research questions of this thesis. The comparative study will consist of finding and defining current IoT communication protocols, new and old.

4.1 Solution alternatives

The solution alternatives are IoT communication protocols and are defined and explained in a chronological order where the oldest protocol is defined first and newest last depending on when the protocol was released and not when it was last updated.

4.1.1 Hypertext Transfer Protocol (1991)

Probably the most known data communication protocol is the Hypertext Transfer Protocol or preferably known as the famous HTTP. Being one of the oldest protocols that is still being used today is one of the factors that contributes to the protocol's fame. Timothy Berners-Lee is a British inventor who when working at CERN¹ developed with his team the current World Wide Web. WWW at that time was a complex system which needed a simple protocol to exchange hypertext documents which was the foundation for the World Wide Web, introduce the birth of HTTP. [10]

HTTP is a protocol that uses the client and server system. The client communicates with the server via requests and the server sends back a response. Being that HTTP was the first communication protocol used for WWW, the responses were usually a web browser of some kind [11]. These requests and responses are being sent over to each other via the secure network player protocol TCP or through a TLS-encrypted TCP connection [11]. Having the communication secure is essential because of the data that is often being sent and received using HTTP. As written earlier, the HTTP requests are usually web browsers, but they could also

be important emails or messages and for that reason, having the communication secure prevents eavesdropping.

In a client-server communication system, the client almost always takes the initiative to send a request but sometimes the server updates or send notifications to the client without the client explicitly requesting them these can often be called Server-sent events (SSE) or using WebSocket's which provides full-duplex communication channels. There are eight types of HTTP requests, and the four most common ones are called GET, POST, PUT, and DELETE. The GET request indicates to the server that the client wants an existing resource from the server, POST is kind of the opposite, it creates a new resource and sends it to the server, PUT requests edits an existing resource and a DELETE request is performed when you want to delete an existing resource. There are also different types of responses that can be obtained from the server, these are called response status code and are represented with numbers. Around 500 different types of status codes exist but the most common ones are "200" which represent that the request was successful or "404" which indicates that the server can not find the requested resource. [12]

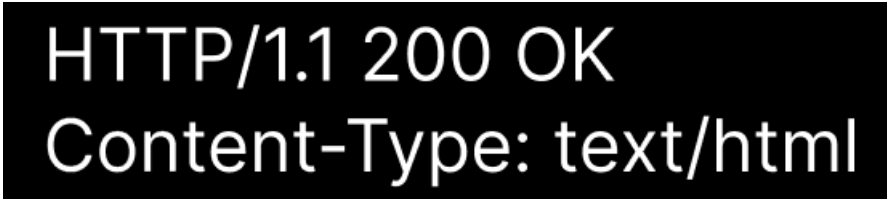
If I wanted to access a web browser let's say <http://www.test.com>. The browser I am using will extract an ip address by taking the domain name, in our case "test.com" and ask the internet domain name server to return the ip address. The client, me and my browser will now connect to a server at that ip address and initialize a GET request which is represented by figure 2 below.



```
GET / HTTP /1.1
Host: www.test.com
```

Figure 2: A request showing what type of request, version of HTTP and the resource trying to obtain.


After receiving the request, the server goes to look for the requested path and if found, a response with the desired resource and status code "200" will be sent back to the browser, see figure 3.



```
HTTP/1.1 200 OK
Content-Type: text/html
```

Figure 3: A successful response showing the version of HTTP, status code, and the resource.

If not found, then the server sends back a response telling the client that the resource is not found by a response status code “404”, see figure 4.



```
HTTP/1.1 404 NOT FOUND
```

Figure 4: An unsuccessful response showing the version of HTTP and the reason behind the error via a response status code.

4.1.2 Message Queue Telemetry Transport (1999)

Message Queue Telemetry Transport or more commonly known for its abbreviation, MQTT. International Business Machines Corporation (IBM) released MQTT in 1999 and was designed to enable monitoring equipment used in the oil and gas sector to communicate data to distant computers. In 2013, the Organization for the Advancement of Structured Information Standards or more preferably, OASIS, standardized MQTT as an open-source protocol and is still, to this day under OASIS’ management. [13]

The name Message Queue Telemetry Transport can be very confusing. MQTT uses a topology called “Publish/Subscribe” and not a message queue even though the name suggests it so to aid the confused, MQTT is also the abbreviation for MQ Telemetry Transport.

Publish/Subscribe is a communication model that utilizes a broker. An MQTT broker controls all data, it manages all the data being sent and it regulates the data so that the receiver gets relevant information [14]. Imagine an office analogy where you are in an office building consisting of several floors. To communicate with your colleagues, you have to use a special code to deliver your message to the correct colleague on the correct floor. The intercom systems act as an MQTT broker, routing your message to the correct person on the correct floor based on the code you entered. With the broker, MQTT clients have no dependencies on each

other, meaning that two clients that communicate with each other will not “know” each other or have to run at the same time. The broker will also eliminate sudden interruptions that could occur under the publishing and receiving part of the communication process. This is called decoupling and is one of the most important aspects of the Publish/Subscribe model. [15]

Publishers and subscribers are terminology used to describe different types of MQTT clients, depending on whether they are actively posting messages or have subscribed to receive them, but this does not limit a client from doing both of these tasks. A subscription occurs when a client receives data from a broker. If the process is performed backwards then it is called a publish. As a client, you subscribe to topics you are interested in via the broker [14]. For example, if I were to use a voice assistant to turn a lamp on in my office, the lamp would be subscribed to the simplified topic, “office/lamp”. If I now publish a message from my voice assistant, saying that I want to turn my office lamp “on” then the broker will update the topic “office/lamp” so that the lamp can receive the intended message, thanks to it being subscribed to the relevant topic. This example is visualized by figure 5 below.

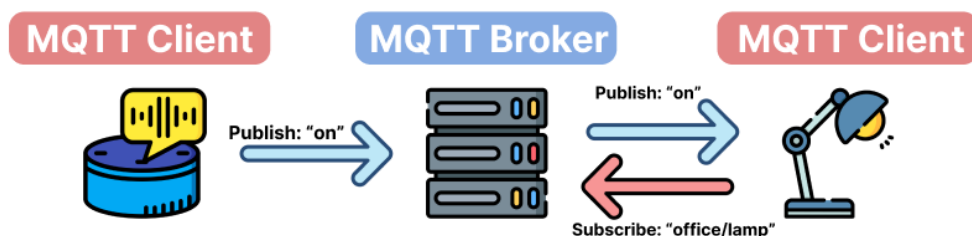


Figure 5: Visualized example of a MQTT communication process.

With the knowledge gained of brokers and the publish/subscribe communication model, we can take a look at how an MQTT session looks like. The session is often divided into four stages: connect, authenticate, communicate, and terminate. In order to connect to the broker, a client first establishes a Transmission Control Protocol/Internet Protocol (TCP/IP) connection. This is usually done by using a normal port that has been set by the broker’s operators, but it could also use a special port provided by the broker’s operators. In some cases, if a client tries to reconnect to the broker, then the disconnected session can be continued. [14]

MQTT relies on the Transmission Control Protocol (TCP) for data transmission. Communication for data transmission can occur as non-encrypted or crypted. To encrypt the conveyed data from interception and alteration, a Secure Sockets Layer (SSL)/Transport Layer Security (TLS) handshake is used. This handshake can be used by the MQTT broker to authenticate the client's identity and vice versa, client authenticating the broker. SSL/TLS may not always be an option and, in some circumstances, may not be desired. These circumstances appear because the MQTT protocol strives to be resource limited meaning that the protocol is designed to be used in environments like low bandwidth and high latency networks. Due to this, a simple username and password can be used for authentication instead. [14]

4.1.3 Extensible Message and Presence Protocol (1999)

Extensible Messaging and Presence Protocol also known as Jabber have the abbreviation, XMPP. This protocol was most frequently used as a messaging protocol, most of the messaging apps started using XMPP like Zoom, Kik Messenger, and WhatsApp. [16]

XMPP is also custom to the client and server communication model but instead of the markup language HTML which is used in HTTP to exchange data between the client and server, XMPP instead uses XML or Extensible Markup Language. As stated earlier, XMPP is used to build chat systems, and for this the criteria or goals to achieve a secure and functional message system is to be able to extend with new features, send one-on-one messages or one-to-many messages and also being able to see if your contacts are online or not.

A client request in XMPP is called a stanza. This refers to an XML element which consists of the fundamentals of an XMPP request such as information about the message being sent, updates or the presence status of a user. More specifically, there are three types of stanzas:

Message Stanza (<message/>): This type of request is used for, as the name suggests, sending a message to a user or a group. The XML element for this stanza is called <message/> and consists of five common attributes. These attributes are more like specifications and guidelines for example, one common attribute is the 'to' attribute which specifies the client id you intend to send the stanza to. Conveniently a 'from' attribute also exists and has the sole purpose of giving the recipient clarification

to who it was that sent the message. [17] An example of how a message stanza could look like, see figure 6 below.

```
<message from="user1@example.com" to="user2@example.com" type="chat">  
  <body>Hello, how are you?</body>  
</message>
```

Figure 6: Example of how a message stanza structure could look like.

Presence Stanza (<presence/>): The presence stanza is used to send online status information and to control subscription status between contacts more like a “publish-subscribe” communication model. To send online status information or data that requires to be broadcasted throughout the remaining clients that connects to a shared server then the presence stanza should be used without a ‘to’ attribute. This will make the server broadcast the presence stanza to the rest of its connected clients. That is why it is possible to see if a user is online or not using the XMPP protocol. [16] Figure 7 below displays how a presence stanza that shows that a user is online could look like.

```
<presence from="user1@example.com" to="user2@example.com">  
  <show>available</show>  
  <status>Online</status>  
</presence>
```

Figure 7: Example of how a presence stanza structure could look like.

IQ (Info/Query): This stanza is used when you want to get information from the server or to set data, replace existing data or provide data for an operation. The IQ stanza basically works in line with the “request-response” communication model. Having said that, a client must specify what type of request needs to be done using the ‘type’ attribute. These could either be a get or a set request. Result and error are types of responses. An id must also be specified which provides a unique identifier for the IQ stanza so when a response occurs, we know what request the response covers. [17] IQ stanzas are very versatile and can be used for a variety of reasons, for example look at figure 8 below showing how a client sends a get request to a server.


```
<iq type="get" id="123456">  
  <query xmlns="jabber:iq:roster"/>  
</iq>
```

Figure 8: Example of how an IQ stanza could look like.

4.1.4 Constrained Application Protocol (2010)

CoAP, stands for Constrained Application Protocol, was created by the Internet Engineering Task Force (IETF) in 2010 to use on IoT devices and machine-to-machine communications. This specialized web transfer protocol was designed to be used in low power and low memory networks, such as IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). Unlike MQTT, CoAP uses a messaging model called "Request/Response". [18]

The Request/Response topology follows a client/server architecture which is very similar to the architecture HTTP follows. One of the most noticeable differences is that a CoAP implementation often performs both client and server responsibilities during a M2M communication. The communication is initialized by the CoAP client, asking for resources from the CoAP server. This is done by sending a Method code. After receiving the code from the client, the CoAP server responds to the action by sending a Response code which could include a presentation of the desired resource. This exchange is handled asynchronously through a datagram-oriented transport protocol, to be specific, User-Datagram protocol (UDP). [18]

Reliability concerns can occur due to the communication being asynchronous but to secure and make the communication more reliable, CoAP provides four types of messages, these are, Confirmable (CON), Non-confirmable (NON), Acknowledgment (ACK), and Reset (RST). A communication process using the constrained application protocol can therefore mark their message as Confirmable to provide reliability. Marking a message Confirmable means that the message will retransmit until the receiver sends an Acknowledgement with the same message id [18]. Reliability may not always be the best approach for M2M communications for example, if an air condition system needs to read repeatedly from a temperature sensor, then it would be more beneficial if the communication did not depend on Confirmable and

Acknowledgment marked messages, in this case the message should be marked with Non-confirmable to indicate that an Acknowledgment is not required. Figure 9 and 10 below shows an example of how these marked messages can be sent and received.

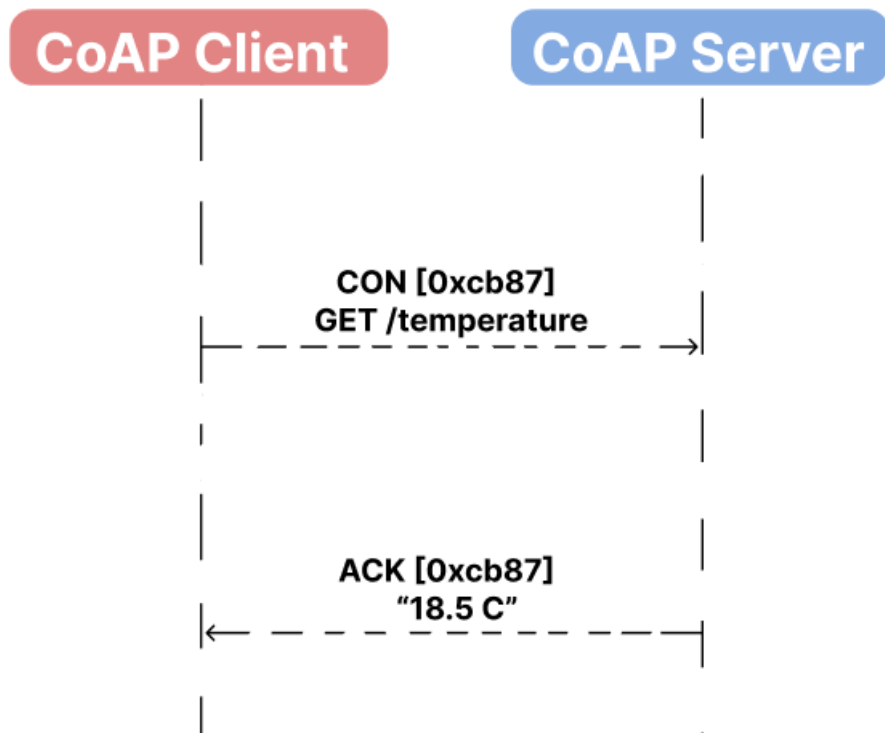


Figure 9: A Confirmed communication, obtaining the temperature.

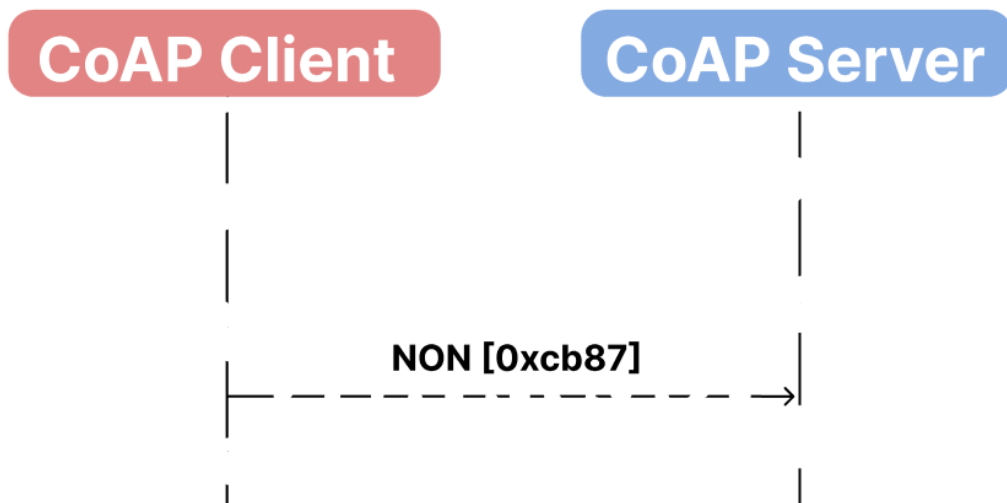


Figure 10: Example of a Non-Confirmed communication.

4.1.5 Matter

This application layered communication protocol intends to revolutionize smart home systems by focusing on interoperability which in the context of matter means that devices of different manufacturers can communicate with each other effectively. The idea is to stop being concerned with always having to find the right device that could support your home system instead find a smart device that supports matter, and you should be worry free.

Matter supports both the network protocols TCP and UDP and is also designed to take advantage of the benefits from IPv6. This application layered communication protocol can be broken down into seven main features showcased by figure 11 below. [19]

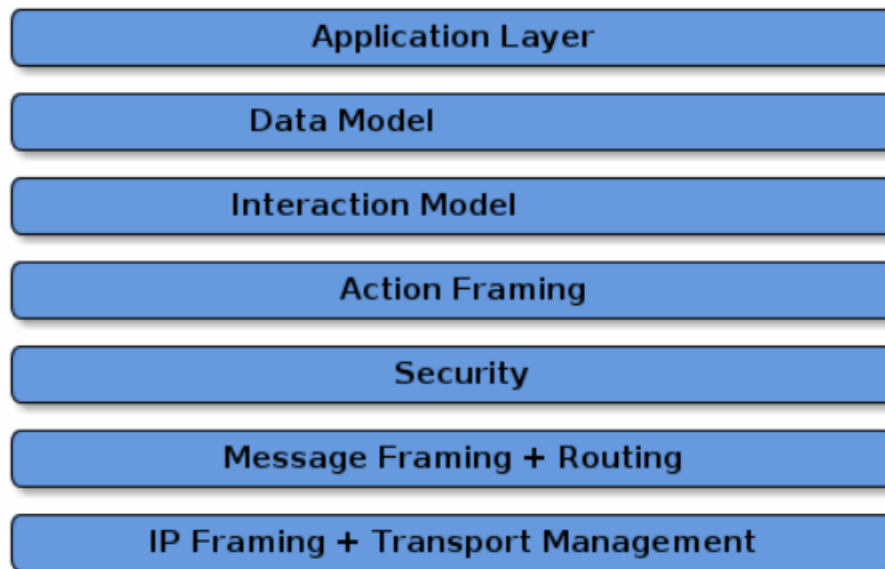


Figure 11: Seven main features of Matter. [19]

Application: This layer corresponds to the high order business logic of a device [19]. For instance, software focused on temperature would probably contain the option to adjust the temperature or even adjusting the temperature based of time or day of week.

Data Model: This layer relates to all data and verb elements that aid in supporting the functionality of the application [19]. In our temperature example above, this data model layer would interact with the application layer by sending data such as current temperature, target temperature and more.

Interaction Model: The third layer counting downwards determines a set of actions that can be conducted between a client and server device. These interactions operate on structures that are defined by the layer above, data model [19]. If a smart matter compatible thermostat is installed in a home, the interaction model would define actions such as requesting temperature information from the thermostat to other devices in the home.

Action Framing: The next layer, action framing is the procedure that serializes the interaction made from the interaction model layer into a predetermined compressed binary format to encode for network transmission. [19]

Security: The security layer has the obvious task to supply some form of security, this is done on the encoded action frame from the action framing layer. The compressed binary format message is encrypted and supplemented with a message authentication code to ensure safety between the transaction from the sender to the receiver. [19]

Message Framing and Routing: At this layer we are left with an action that is both serialized and encrypted thanks to the layer above. What this layer does is construct the payload format of the message. The payload consists of required and optional header fields. The properties of the message are therefore specified by the payload. This layer also handles logical routing information. [19]

IP Framing and Transport Management: The last layer of the encapsulation process involves sending the fully crafted message to the underlying transport protocol which is either TCP or UDP depending on certain circumstances, where the message will be IP managed. [19]

Once the encapsulation is done, the data will be sent to the receiver and when it arrives, the process of decapsulation begins, where the various layers reverse the operations on the data given by the sender. So, the message will start from the IP Framing and Transport Management layer and go upwards until it reaches the application layer where the data finally gets consumed.

4.2 Comparison of solutions

A comparison table is used to make a clear comparison between the protocols so choosing what protocol to test with will be easier. The table will consist of several metrics that include a specific specification. These specifications are communication method, which describe what the underlying protocol uses for communication method to transport messages. This could affect the tests by having different message sending processes. Next is what network protocol each IoT protocol uses, a slow and secure network protocol would probably give different measurements to a fast and unsecure network protocol. Some protocols are not open source which limits the testing one can do. QoS or quality of service is a message delivery guarantee that the protocols who support QoS provide which is also important for testing to assure that every message gets delivered and no sudden disruptions of the measurement can occur. A message can be built differently and with that, a limit cap applies to the payload size which could affect the way a test has to be constructed because having a limited size would restrict the amount of data that can be transmitted in a single message. The comparison table is showcased by table 1 below.

	Communication method	Network Protocol	Open Source	QoS	Payload Size Limit
HTTP	Client-Server	TCP	Yes	No	Unlimited
MQTT	Publish-Subscribe	TCP	Yes	Yes	Unlimited
XMPP	Client-Server (Instant messaging)	TCP	Yes	No	64 KB
CoAP	Client-Server	UDP	Yes	Yes	1280 Bytes
Matter	Client-Server	TCP/UDP	Yes	Yes	Unlimited

Table 1: Comparison table for each protocol studied.

4.3 Chosen solution

I have chosen to test a limited amount of the protocols mentioned above, more specifically three protocols. Two more prominent protocols and one new.

The communication protocols that will be further investigated are Constrained Application Protocol (CoAP), Message Queue Telemetry

Transport (MQTT) and Matter. The Matter protocol will also be the protocol I will create an open-source client with. These protocols are chosen because each protocol suits the idea of having three different protocols while being very testable. CoAP and Matter use a client-server communication method which differs from MQTT that instead uses a publish-subscribe model, this would give an insight into how the separate methods could affect the tests and conclude what model is the more superior method to what specific area. Choosing CoAP and Matter over the other client-server protocols is because CoAP are heavily dependent on UDP and Matter uses both depending on the request. Having protocols that utilize different network protocols would also give an understanding of what network protocol will give the faster measurements or reliably safe delivery, so no message/request gets lost. Relevancy also affected the chosen protocols, XMPP is not as commonly used on apps as it was when it first got released which resulted in excluding the option to choose it. HTTP on the other hand is very relevant and is still used today but the reason behind it being excluded is the protocols disregard to QoS meaning that a message could potentially not be delivered which would damage the tests and to eliminate that risk, I have chosen to not bother with HTTP.

Matter is a new developed IoT protocols that have very limited documentation, so picking Matter as the protocol to research and creating an open-source client with was very obvious for me, because the company behind Matter have gained a lot of attention for its products and the capabilities of Matter so researching what could possibly be the future for smart home devices is necessary.

5 Implementation

This chapter provides a technical overview of how the implementation was built and how each protocol was implemented. A system overview can be seen in Figure 12, where each protocol and their respective libraries are shown. The arrows indicate how the sending and receiving process works for each protocol. The measurement implementation is indicated with a timer logo that represents where the timer will start and end, to measure how long the process takes. Each device is described in their respective subsections below.

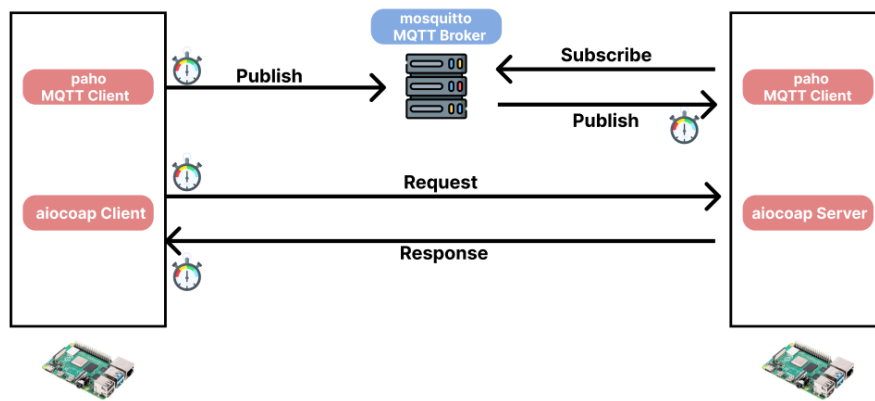


Figure 12: System overview.

5.1 Message Queue Telemetry Transport

The Message Queue Telemetry Transport implementation is done using the Python language and utilizing the *paho-mqtt* library as the main program library. The *paho-mqtt* library provides functionality to create MQTT clients and make them interact with MQTT brokers. The implementation starts with creating two MQTT client instances and defining how it should act when doing a publish or subscribing. The clients should then both connect to the broker via a method provided by the *paho-mqtt* library.

The implementation differs between the clients, depending on what purpose it has and what performance factor is measured. Refer to appendix A for both the MQTT publish and subscribe client code.

Measuring latency with a protocol that is not using the straightforward client-server communication method can be a bit tricky, Because in a

traditional request-response models, measuring performance such as latency is only handled by the same client since it is the one requesting and receiving but, because of the fact that MQTT uses a publish-subscribe model and have a broker controlling and regulating the communication and data, it would be quite difficult to measure the time it takes for a message to travel to the recipient client since the communication method inherently involves two separate client implementations. This separation adds complexity to the process of measuring performance.

5.1.1 MQTT Publish Client

The MQTT client responsible for publishing messages imports these required libraries: The *paho-mqtt*, *time*, and *msgpack* library. The *paho-mqtt* is crucial for implementing the MQTT client, while the *time* module is used for time-related tasks such as in this case, measuring the time it takes to send and receive a message. The *msgpack* library will allow for efficient and compact data exchange so that the encoding and decoding part of the message will not affect the measurements much.

As mentioned in chapter 5.1, measuring performance on MQTT can be a complex task, so what I have done is incorporate a timestamp in the published message. This is done by utilizing the MessagePack library which encodes data in a compact binary format, so a lesser data size is achieved which in turn allows for a faster data transmission. Sending the timestamp via the message is done for multiple of reasons one of them is simplifying the latency calculation, by giving the subscriber easy access to the initial publishing timestamp. This method also prevents the need for explicit communication between the publisher and subscriber client, as this data is carried within the message payload. Another reason is that it reduces the impact of clock synchronization errors. Having a precise clock synchronization system would be the optimal choice, but having a complex system where a broker and two different clients are involved could introduce errors and inaccuracies in the latency and throughput measurements. Including the timestamp in the published message allows for flexibility and aids the measurement for scalability because let's say we have multiple subscribers waiting patiently for the same message, they can each calculate their respective latencies using the same timestamp provided by the publisher.

Regulating the number of messages to be sent is essential for measurements and is done by a variable holding the value referring to

the number of messages that will be sent. To get some feedback that a message has been sent successfully, a function called *“published”* will be called upon whenever the client instance makes a publish. This function will then print out a message indicating that the publish was successful to the terminal. This is done by setting the member function *“on_public”* from the client instance to the *“published”* function and it will then work as a call back function for the client.

The client connects to the MQTT broker running on the local machine (“localhost”) at the default port number 1883 which is assigned by the Internet Assigned Numbers Authority (IANA) [20], with a 60-second keep alive interval, which means that the connection between the MQTT client and MQTT broker is “kept-alive” or active, 60 represents the maximum time allowed between two consecutive exchanged messages. This reduces resource usage such as network traffic so bandwidth could be conserved. It also detects disconnections which is useful when doing a performance measure because it will ensure accurate measurement for the simple reason that when a disconnect during the test occurs, errors in the measurement data could be present. Detecting disconnections could also identify bottlenecks and hidden issues, for example if a disconnect occurs frequently during periods of high message throughput, could indicate that a problem with the MQTT broker might be the case, so finding these hidden issues would give me the chance to improve and optimize my little homemade MQTT system. If a client does not send a control packet, such as a publish, subscribe or unsubscribe within the keep-alive interval then the broker assumes that the connection with the client is lost and proceeds to close the connection with the client. [21]

Now that the client is connected to the broker, it can finally start its network loop, allowing it to process network events such as sending and receiving messages. This loop runs in the background while the rest of the program is executed.

Finally, we can start publishing our messages, to do that we need to first locate where we aim to send them. This is done by choosing a topic, which is where the subscribing client will be subscribed to. Then because of the measurements, we start our timer. This timer is being published with the message as a part of it. All this process is done iteratively using a *for I in range* loop where the range is set to the variable declaring how many messages will be sent. See figure 13 and 14 for a visualization of a

flow chart that simplifies the overview of how the publishing client measures latency and throughput.

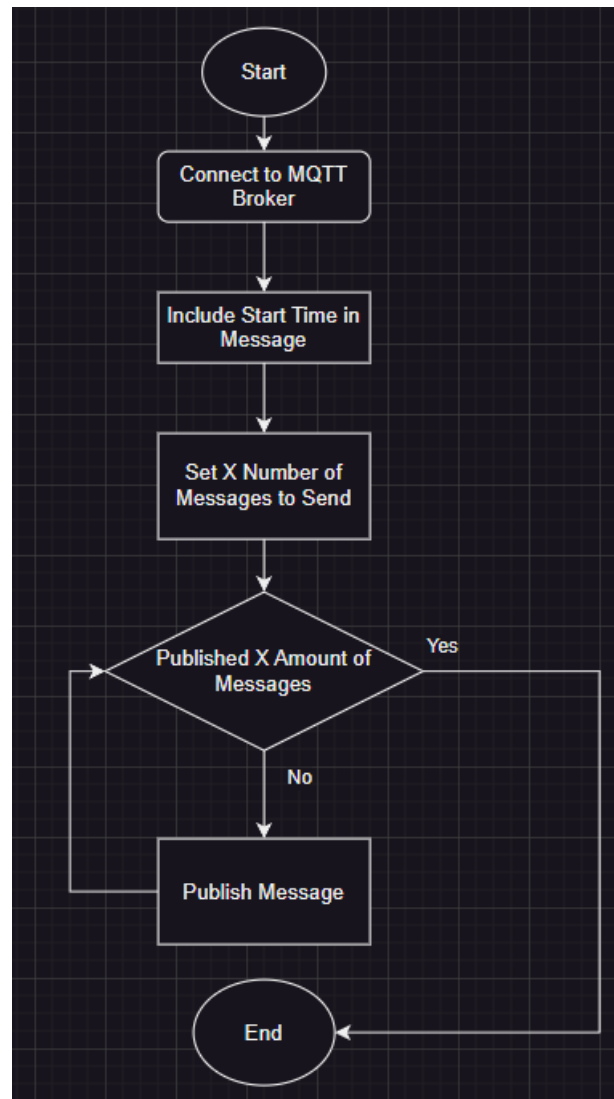


Figure 13: Flowchart representing how the Publishing client works when measuring latency.

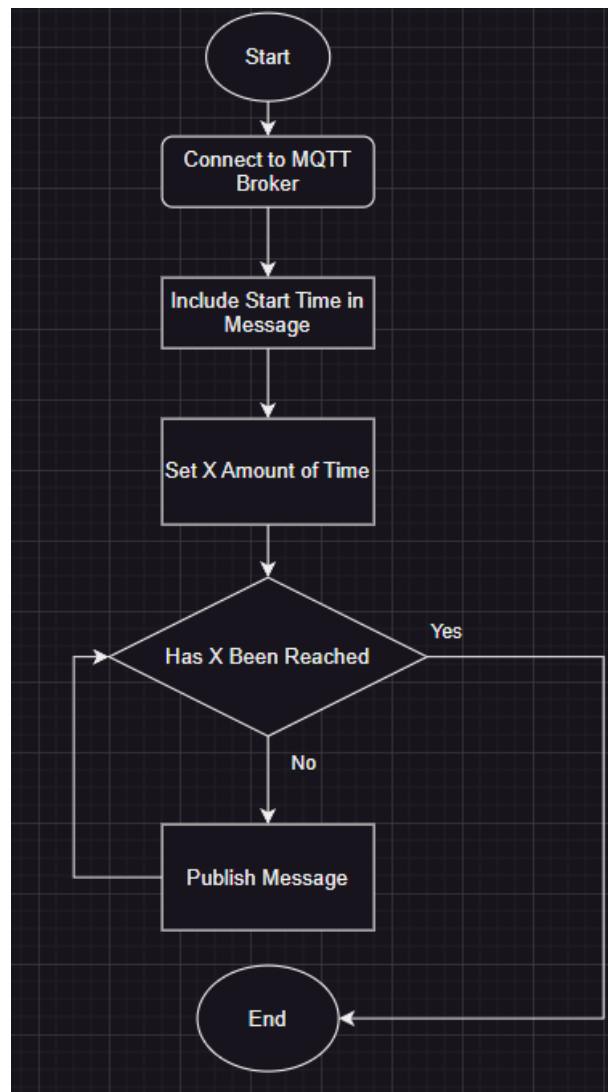


Figure 14: Flowchart representing how the Publishing client works when measuring throughput.

5.1.2 MQTT Subscribe Client

The same imported libraries for the publishing MQTT client are also present through each implementation of the MQTT subscribe client. The MQTT subscribing client is built in a similar way as the publishing clients in aspect of creating an MQTT client instance and connecting to the broker as well as starting its execution loop or more formally, actively subscribing and waiting for publishes. However, there are some huge differences.

To receive and do computations with the time from the published messages, a defined callback function is made where it will be called

whenever a message is received from the subscribed topic. When a message is received, it calculates the time elapsed since the message was sent, decodes the received message using MessagePack, and stores the elapsed time to a file. MessagePack is advantageous for the outcome of the measurements because it is an efficient binary serialization format that helps reduce data size and processing time, which reduces the time taken to deserialize messages which in turn lessens the overall elapsed time for the message, so the measurement is as dependent on the MQTT sending and receiving process as it can be.

When measuring the throughput, a stream of finite number of requests are being timed to see how many messages can be sent per second. This is done by declaring a variable that consists of the number of requests that shall be made before stopping the MQTT subscribing client from receiving any more publishes. An infinite while loop keeps the MQTT client active until the desired request amount is received. A monotonic timer which is a clock that never goes backward even if the system time is modified. System time is also known as wall-clock time and is the clock that is maintained by a computer's operating system. A monotonic clock is used because of its reliability, since it only moves forward, the elapsed time will always be consistent and reliable providing with accurate calculations. This timer is placed right before the infinite loop and one at the end which will be executed right after the subscribing client receives the desired number of requests. Calculating the total time to receive all messages is next and with that value the total throughput, or message per second can be computed.

Finally, the throughput value is stored in a file for further testing. See figure 15 for a flow chart that describes how the MQTT subscribing client operates.

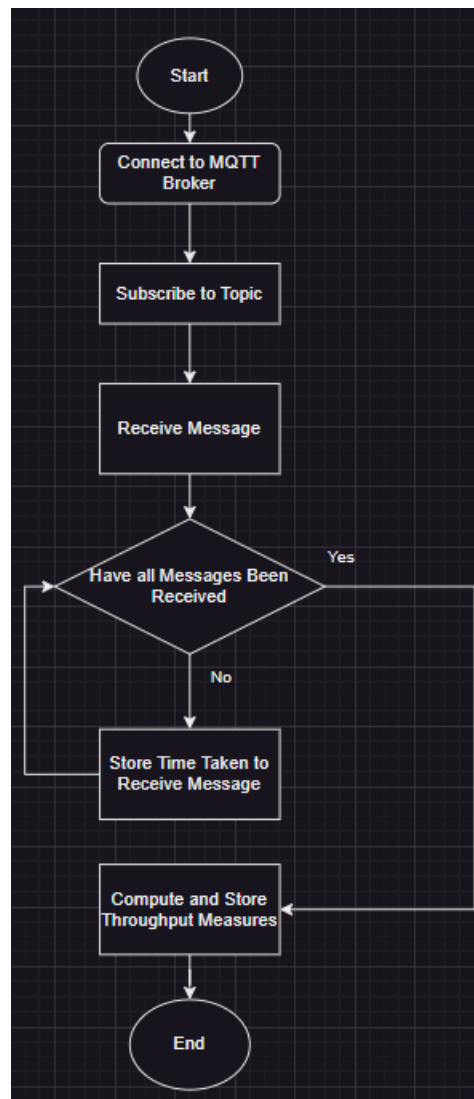


Figure 15: Flowchart representing how the Subscribing client works.

5.1.3 MQTT Broker

The broker used for testing and connecting the clients is a local Mosquitto MQTT broker. Mosquitto is an open-source MQTT broker developed by the Eclipse Foundation. [22]

This broker is responsible for receiving, filtering, and distributing messages between clients that publish messages to topics and clients subscribing to topics. This broker is running all the time in the background, waiting patiently for clients to connect with it.

5.2 Constrained Application Protocol

Implementing the Constrained Application IoT protocol is done using the Python language with the *aiocoap* library as the main program library. The *aiocoap* library provides a full-featured framework for building CoAP-based applications, it supplies the user with the functionality to create CoAP clients and servers, which makes the tests much easier because both client and server can be used by the same framework. Refer to appendix B for the CoAP client source code.

5.2.1 CoAP Client

The MQTT client responsible for publishing messages imports these required libraries: The *paho-mqtt*, *time*, and *msgpack* library. The *paho-mqtt* is crucial for implementing the MQTT client, while the *time* module is used for time-related tasks such as in this case, measuring the time it takes to send and receive a message. The *msgpack* library will allow for efficient and compact data exchange so that the encoding and decoding part of the message will not affect the measurements much.

Implementing the CoAP client requires these libraries: The *asyncio*, *time*, and *aiocoap* library. The *aiocoap* library is obviously crucial for implementing the CoAP client and the *time* library is used for time-related tasks such as in this case, measuring the time it takes to send and receive a message. The implementation uses an asynchronous way of sending requests meaning that each request will not have to wait for a response so that it can send another request which is done in a synchronous system. This is done because of the fact that CoAP utilizes the network protocol UDP, which is non-blocking meaning that when a request is sent, the initial sender will not “sit” and wait for a response from the receiver, instead it will just continue doing what it was doing before, whether it is sending another request or updating data. So, for that reason, an asynchronous implementation of sending requests is far more accurate to the way a CoAP client should act.

The asynchronous system works in the way of managing control and suspending certain tasks. The python expression *await* is used throughout the implementation to handle the sending and receiving process asynchronously. This is used when sending a request, the request will be sent to the server, meanwhile the client will still execute the rest of the program which would proceed to send another request. When an awaited request gets a response, the program will resolve the *await* expression and would continue from the original state.

The CoAP client implementation measures the latency and throughput using the same code, first a variable containing the desired number of requests that should occur gets declared. A for loop iterating until the number of wanted requests is made. Two monotonic timers are used, one to measure the throughput which is placed right before the for loop is initialized and one right before a request is made. The elapsed latency time is stored in a file and when the for loop is done executing, a measurement for throughput is made and stored in a separate file. See figure 16 for a simplified overview of the implementation of the CoAP client by using a flow chart.

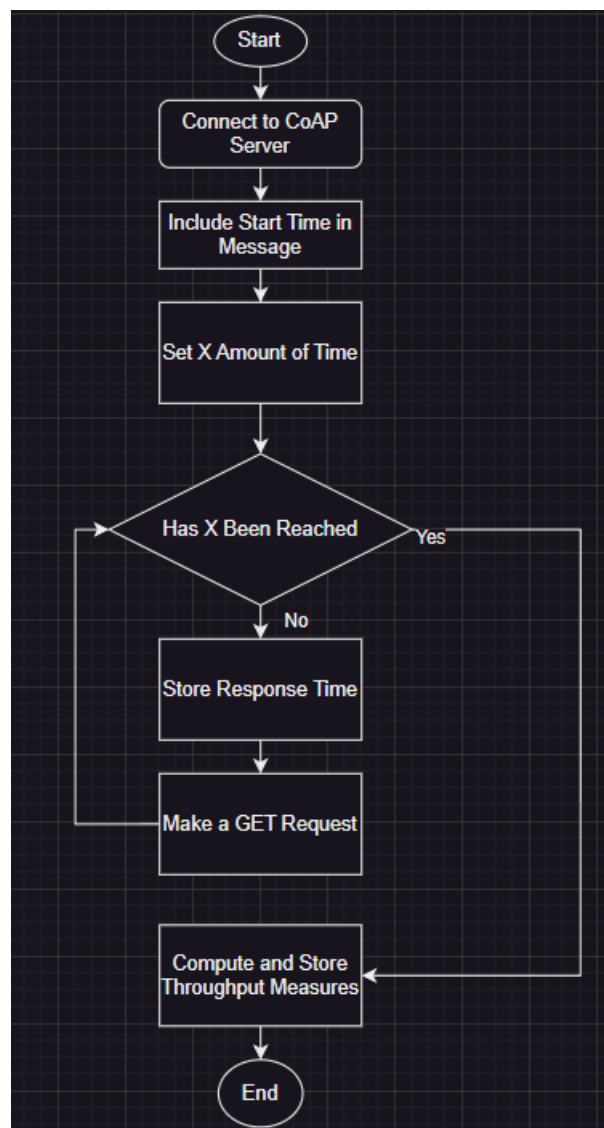


Figure 16: Flowchart representing how the CoAP client works.

5.2.2 CoAP Server

The implementation of the CoAP server is heavily influenced by the reference implementation provided by aiocoap website [23]. The imported libraries include the ones contained in the CoAP client and the *tracemalloc* and *datetime* libraries. Tracemalloc memory profiler starts at the beginning and has the objective to find memory leaks and to optimize memory usage. Because of this, handling a large number of requests will not have a huge impact on the server.

BlockResource is a class defined next which supports two types of requests, namely GET and PUT requests. A different approach of action is done depending on the type of request, if it is a GET request then the server will return a response. This triggers a transfer called blockwise transfer, which is when the response is sent in multiple packets that are called blocks. This is necessary because the response being sent back is going to be larger than 1024 bytes and sending it back in multiple blocks will improve reliability for the simple fact that if one block is lost or corrupted during transmission then only one block needs to be resent. When a PUT request is received from a client then the desired update or replace order will be placed on the PUT request payload, so the BlockResource class will only update and replace the resource with the payload from the request.

5.3 Matter

A matter system could not be built and tested with for several reasons. First, there is very limited documentation for the Matter protocol since it is so new. The Matter specification is still not fully public, only parts of it have been released which makes it very difficult to build a fully compliant Matter client or server.

Researching Matter and trying to find a library that provided the required functionality to build a client and or server was not only hard but very confusing because, most of the available python libraries with similar name as Matter were libraries that supported functionality for a messaging app called Mattermost. Libraries like *matterapi* [24] and *pymatter* [25] have very limited information about the contents which could confuse a Matter researcher with their names.

There were sites that did offer Matter implementations, but they required some form of a smart hub which had to support Matter, such as the Google Nest Hub (2nd generation) [26]. Even the Connected Home over IP

(CHIP) which is the older version of Matter, have a GitHub repository [27]. This repository only contained partial specifications likewise to the Matter core specification [19], there are also no tools in the CHIP repo to thoroughly test, validate, and certify a new Matter client or server and these types of tools are essential to ensure any Matter implementation meets the core specification so authentic measurements can be obtained.

With that said, the majority of any form of “homemade” Matter implementation is only designed to communicate with smart home Matter compatible devices so a customization is available, such as programming a Matter supported light switch to turn on every time someone enters a room or returns home, and with having time and resource limits, communicating with a Matter compatible smart home device were unfortunately not an option.

5.4 Evaluation setup

(All the measurements taken from the test for the two protocols are stored in separate files, these files are then used to evaluate the values by visualizing them in a bar chart format. Using MATLAB, a script is done that takes the content of the file and computes the mean and standard deviation. These values will now be inputted to a bar chart format which the MATLAB script generates and outputs a customized bar chart containing the tests mean and standard deviation. See appendix C for the MATLAB code.

6 Results

This chapter presents the results of testing the performance of three IoT communication protocols in terms of latency, throughput, and scalability. These results include tables with the mean value and standard deviation for each performance factor measured on each protocol. The tests were conducted in a controlled environment using the same hardware and software configurations for each protocol. To prevent any interference from the network or other system resources so an optimal level performance can be achieved, the tests were conducted with no other applications running in the background, except for the clients and servers being tested.

6.1 Resulting system

Figure 17 shows what the terminal will output when the server is ready to receive requests.

```
PS C:\Users\46729\aiocoap> python server.py
INFO:websockets.server:server listening on 127.0.0.1:8683
INFO:websockets.server:server listening on [::1]:8683
DEBUG:coap-server:Server ready to receive requests
```

Figure 17: Example output of running the CoAP server.

Running the client that sends the request will show the time it took to send and receive the request/response and if it was successful, as shown in Figure 18.

```
Elapsed Time: 5.01 ms
Result: 2.05 Content
b'2023-05-19 23:42'
Elapsed Time: 4.79 ms
Result: 2.05 Content
b'2023-05-19 23:42'
```

Figure 18: Example output of running the CoAP client.

Connecting and subscribing to a topic for the CoAP subscribing client will output a message to clarify that the connection was successful, as shown in figure 19.

```
PS C:\Users\46729\paho.mqtt.python\tests> python clientSub.py
Connected to MQTT broker
```

Figure 19: Example output of running the MQTT Subscribing client.

Figure 20 shows information about the status of the published message done by the MQTT publishing client.

```
PS C:\Users\46729\paho.mqtt.python\tests> python clientPub.py
Message 1 published successfully
Message 2 published successfully
Message 3 published successfully
Message 4 published successfully
Message 5 published successfully
```

Figure 20: Example output of running the MQTT Publishing client.

6.2 Measurement results

Every measurement made is showcased in this chapter where the measurements are categorized in their respective performance factor, allowing for an in-depth understanding of how each factor contributes to the overall protocol performance.

6.2.1 Latency

For a better understanding of latency, please refer to chapter 2.4.1. Every latency measurement was made to have the same test conditions for each of the three IoT communication protocols.

The latency was measured by sending a series of requests/publishes from a client to another client or broker, depending on what protocol is being tested. The series consists of three tests where each test includes a different amount of request/publishes. This approach was taken to ensure precise reading and to identify any potential fluctuations. A bar chart consisting of each protocol measurement will be shown at the end of every series, see figure 21. Table 2, 3 and 4 show the mean and standard deviation measurements for each respective series.

	Mean (ms)	Stdev (ms)
CoAP	3.97123	0.76441
MQTT	3.25412	0.68960

Table 2: Measurements of 100 requests/publishes.

	Mean	Stdev
--	------	-------

	(ms)	(ms)
CoAP	3.82674	0.89397
MQTT	3.28753	0.72541

Table 3: Measurements of 500 requests/publishes.

	Mean (ms)	Stdev (ms)
CoAP	3.86822	0.74210
MQTT	3.20921	0.70213

Table 4: Measurements of 1000 requests/publishes.

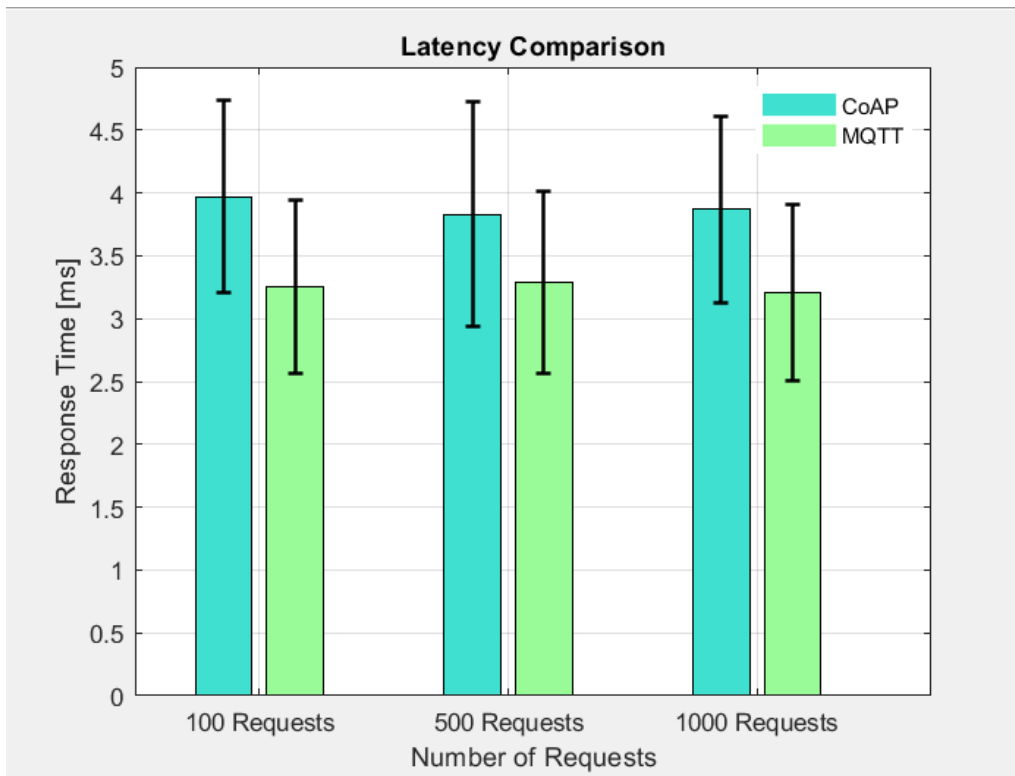


Figure 21: Bar chart showing latency comparisons.

6.2.2 Throughput

Refer to chapter 2.4.2 to gain a deeper understanding of throughput. Just as the measurements for latency, every throughput measurement was made to have the same test conditions for each of the three IoT communication protocols.

To measure the throughput, a continuous stream of messages was sent until a specified time frame was reached to determine the number of

messages per second that the protocol can achieve. Three tests were made for each protocol where each test concluded different time frame. This is done so a more accurate discussion can be had because of more existing data. A bar chart consisting of each protocol measurement will be shown at the end of every series, see figure 22. Table 5, 6 and 7 show the mean and standard deviation measurements for each respective series.

	Mean (msg/s)	Stdev (msg/s)
CoAP	256.41025	2.32468
MQTT	301.03159	3.01269

Table 5: Measurements of 10 seconds time frame.

	Mean (msg/s)	Stdev (msg/s)
CoAP	253.15883	2.74814
MQTT	299.92302	2.80297

Table 6: Measurements of 30 seconds time frame.

	Mean (msg/s)	Stdev (msg/s)
CoAP	254.13480	2.71002
MQTT	300.35590	2.79713

Table 7: Measurements of 60 seconds time frame.

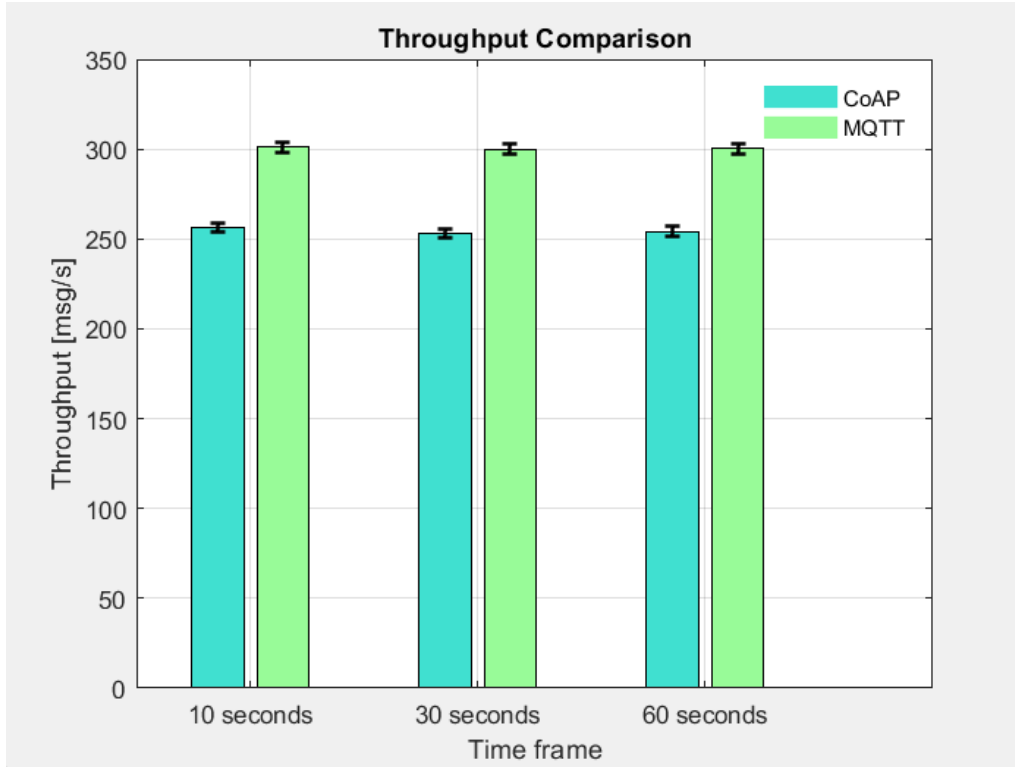


Figure 22: Bar chart showing throughput comparisons.

6.2.3 Scalability

Refer to chapter 2.4.3 for a more comprehensive understanding of scalability. Similar to the measurements for latency and throughput, every scalability measurement was taken under the same test conditions for each of the three IoT communication protocols.

Measuring scalability has some similarities to this thesis measurements for latency. Three different amounts of devices/terminals will be active during the testing for each protocol. With every test series, a different number of request/publishes will be made to get a broaden understanding of how different amounts of devices/terminals can affect the latency. Table 8, 9 and 10 show the mean, standard deviation, and throughput measurements for each respective series.

	Mean (ms)	Stdev (ms)	Throughput (msg/s)
CoAP	4.01289	0.68255	249.19782
MQTT	3.33124	0.72422	300.18843

Table 8: Measurements of 2 clients simultaneously.

Having two clients simultaneously running and requesting has changed the latency for each protocol. An increase of 3.74% from the CoAP latency measure with the lowest standard deviation and an increase of 2.37% for the MQTT latency value with the lowest standard deviation.

	Mean (ms)	Stdev (ms)	Throughput (msg/s)
CoAP	4.17458	0.66982	239.55478
MQTT	3.45327	0.69550	289.58175

Table 9: Measurements of 5 clients simultaneously.

Having five clients simultaneously running and requesting has changed the latency for each protocol. An increase of 4.03% from the CoAP measure with two clients simultaneously and an increase of 3.66% for the MQTT measure with the two clients simultaneously.

	Mean (ms)	Stdev (ms)	Throughput (msg/s)
CoAP	4.45893	0.74244	225.36721
MQTT	3.65772	0.73115	273.07842

Table 10: Measurements of 10 clients simultaneously.

Having ten clients simultaneously running and requesting has changed the latency for each protocol. An increase of 6.81% from the CoAP measure with five clients simultaneously and an increase of 5.92% for the MQTT measure with the five clients simultaneously.

7 Discussion

In this chapter, we embark on a thorough analysis and discussion of the results obtained throughout the course of this thesis. Critically examine the chosen methodology and approaches, exploring the reasons for choosing them and evaluating their effectiveness in addressing the research questions. Furthermore, a discussion about the ethical and societal takes will be presented.

7.1 Analysis and discussion of results

My tests show that MQTT is a bit faster than CoAP in metrics such as latency, throughput and it also scales better. This could seem quite odd if you factor in that CoAP uses UDP as its primary network protocol while MQTT used TCP. UDP is generally the faster protocol and should therefore in practice generate faster results in metrics like latency. Having said that, there are several factors that can affect the result that make the outcome I have obtained justified. Starting off with the base line, the libraries used. The imitated libraries may not fully replicate the performance characteristics of the protocol it imitates which could introduce excessive processing overhead that in turn delays the sending and receiving process a tad bit. Not achieving a fully imitated protocol implementation from the library comes with having a lack of adherence for the protocol's specifications. This absence could result in messages being incorrectly handled, such as not accurately handling QoS (Quality of Service) levels of the message or resource management. Higher QoS levels introduce additional message exchanges which leads to an increased latency measurement. I intended to implement MQTT with a QoS level of 0, which is the equivalent of just sending a message and forgetting about it. This was done for the simple reason that I wanted to see if MQTT could compete with CoAP if the publishing process had a similar behavior to CoAP's UDP utilization, and as the result shows, it can.

Another reason for getting the results is also influenced by the network protocol of both MQTT and CoAP. For TCP, a single TCP connection must be made for all messages between a client and a broker. This means that before I start publishing or waiting to receive messages, I only have to connect to the broker once but, when it comes to sending requests with UDP, each request establishes a new UDP transaction. This leads to a

higher connection overhead if compared to MQTT and in turn would present a higher latency measurement.

I measured throughput in message per second instead of bytes per second which it also could be measured with. Bytes would then refer to the amount of data being transmitted, but in my results, I have limited the throughput to just measure the amount of useful data being sent and not including unnecessary overhead. If I instead used Wireshark to track and analysis each packet sent and then measure throughput, then I can say with certainty that MQTT would have a lower throughput than CoAP. This is because MQTT uses TCP and would do a Three-Way handshake and sends a lot of ACKS (Acknowledgements) which would pile a sizeable overhead and yield a lower throughput (bps) compared to CoAP.

The programming language used for each implementation could also have an impact which is also one of the reasons why both libraries for CoAP and MQTT are Python based. Programming languages have different performance characteristics and are optimized for a variety of areas. If one protocol was implemented using for example, C++ and the other Python then it would highly be likely to see some affect in the measurement. Influencing the development speed, concurrency is something that the choice of programming language has an effect upon and is therefore something to consider when discussing the results.

Even though testing Matter was not an option, I could speculate with some minor degree of certainty how the Matter protocol would compete with the other two protocols. Matter is designed with similar reasons to CoAP, such as being optimized for low-power, wireless IoT devices. Both protocols are built IP and 6LoWPAN standards which enables efficient transmission. Matter uses IPv6 for its operational communications where CoAP uses IPv4. Providing either TCP or UDP transmissions with Matter is also something different from CoAP. This would lead me to believe that Matter would be slightly lower than CoAP depending on how the Matter implementation is built, if it is built that utilizes TCP then yes, I would certainly say that CoAP would generate faster latency measurements, but with UDP as a primary network protocol for the implementation of Matter, the measurements would be closer. So, in summary if I could speculate, I would say that Matter is a bit faster than MQTT but a bit slower than CoAP. This speculation is based on the

general idea that CoAP is faster than MQTT because of several reasons, where the main one probably is that CoAP uses UDP and MQTT uses TCP. Based on the tests accumulated in this thesis, I would say that Matter probably would generate a slower latency measurement than both CoAP and MQTT. This is because the results show a very low margin between the mean values of both protocols which leads to a larger margin where Matter could potentially be either slower or faster, and having a complex system like Matter, the chances in my opinion of being faster is slimmer than being slower.

7.2 Project method discussion

Starting off with a problem formulation has been a good foundation for understanding the question I am researching by clearly defining the problem and its requirements to be solved. Gaining knowledge before implementing has really provided me with focus and guidance for the rest of the methodology. Problem formulation could also reveal solutions that one might not have considered by introducing different perspectives. I would say that just doing research on each protocol before choosing what protocol to take for the next phase is not enough. This is because solely understanding how the protocol works will not be sufficient enough to make the rational decision of choosing to stick with the protocol throughout the rest of the phases, instead I should study and explore if the next phase is doable with the protocol, which in my case would be to find if there existed suitable options to implement the protocol. This became a problem for me because Matter did not have a satisfactory option of implementation and since it was the protocol, I tested last, made it very difficult to change protocol to a new one.

7.3 Scientific discussion

IoT devices only grow in popularity with the years going by, each day a smart device keeps on getting “smarter”. Communicating with these devices is an essential part in their way of functioning in a smart way and should therefore research the communication IoT protocols to get a better understanding of which is better and why. The scientific knowledge gained in this thesis is that certain communication protocols are more suited for a variety of areas. For example, CoAP is designed to for resource-constrained devices which is part of the reason why CoAP uses UDP as its primary network protocol so it can achieve a low overhead which is beneficial for low resource dependent devices. Factoring in the

communication method is also a crucial part of understanding which protocol to be used, such as an application that frequently requires efficiently distributing messages to many receivers, then MQTT would be a great choice because of the publish/subscribe model it uses.

This thesis also gained the scientific knowledge of really learning new and upcoming protocols that could potentially be the new standard for IoT devices going forward in their respective areas, whether the learning part being a quantitative or a qualitative study. Learning about the new protocols is good for a couple of reasons, one is to prepare for the future. You never know what communication protocol will be used in five or ten years span so, learning about them now even if they are not fully documented like Matter, will give an insight into what similarities and differences there are with the newer and the older more used protocols. The research gained about Matter in this thesis could serve as a basis for other people researching the protocols or companies in their assessment of the technology that is included with the Matter protocol.

There are some similarities between my thesis and the related work that are brought up in chapter 2.5, such as doing a comparison between the CoAP and MQTT protocols. The way we did our comparison differs a bit where one did a comparative study which is based on the protocol's characteristic where the other studied some performance factors, mainly latency and CPU usage. The biggest difference is that I did not only study latency but also throughput and scalability for CoAP and MQTT. This thesis also includes a study on the Matter protocol and how it works.

7.4 Ethical and societal discussion

If a large global system that used CoAP or MQTT is built, where a lot of requests/publishes will have to be made would in turn lead to a higher energy consumption globally, which thereby could have an environmental impact such as, accelerate climate change that could worsen the issues of air and water pollution.

Communication protocols like the ones that were tested today, form the foundation for connecting smart IoT devices and exchanging data between devices. This shared information could contain sensitive and private data about individuals or organizations. So, it is crucial that communication protocols provide strong security so that the information is safe from unauthorized access.

8 Conclusions

The aim of this project was to learn about new protocols and how different they are to other more adapted protocols and in conclusion, when it comes to the performance aspect, I would say that there exists insufficient documentation that can steer one into making an implementation to test the performance.

It can be difficult to get an accurate description of the performance when measuring a protocol because of the number of factors that play and could affect the measurements. In theory, CoAP should generate faster latency estimates but as seen from the test done in this thesis, MQTT manages to edge CoAP a bit when it comes to measuring performance due to multiple of reasons discussed in chapter 7.1.

Lastly, research question three for this thesis was to examine if the documentation of a newer protocol which in this case would be Matter, would be enough to create an open-source client with which in turn would be compared to the documentation that exists for the more established and older communications protocols. The implementation of this was never executed due to time constraints which means that research question three was unfortunately never answered in full.

8.1 Future Work

For future work, the current solution would be to expand the research of the Matter protocol, whether it is by exploring deeper into its specification or trying to set up some client that interacts with smart home devices. A deeper comparison between MQTT and CoAP could also be made, where more metrics can be explored such as reliability, interoperability, and security.

8.1.1 Deep Dive into the Matter Protocol

Continuous research on the Matter protocol with this thesis as basis would be an example of a future work. In addition, creating a matter client and a server of some sort that can connect with each other so that the performance of the protocol can be measured. If creating a client and server is not available, then designing by customizing actions on a Matter compatible smart home device could also work, where finding a way to measure some metric would be favorable.

Overall, the significance of this work lies in its potential to discover more about the newly established protocol, Matter. The implementation of Matter will give insight to how the protocol performs and if it is a valid option in the future to be a standard communication protocol.

8.1.2 Further Performance Comparison Between CoAP and MQTT

This project only measured latency, throughput, and scalability as performance factors for both CoAP and MQTT. These three are not the only performance factors that exist and for a deeper comparison when it comes to performance, other factors such as interoperability, reliability, and security are a few options to look further into and examine with.

This example of future work based of this thesis would give a deeper understanding of how MQTT and CoAP work and further draw conclusions to why one of the protocols might be more suitable than the other for certain applications.

References

- [1] Madakam S, Ramaswamy R, Tripathi S. Internet of Things (IoT): A Literature Review. Journal of Computer and Communications [Internet]. 2015;03(05):164–73. Available from: https://file.scirp.org/pdf/JCC_2015052516013923.pdf
- [2] Gillis A. What is iot (internet of things) and how does it work? [Internet]. IoT Agenda. 2022. Available from: <https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT>
- [3] ITU-T. 2012 Jun. Available from: <https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=y.2060>
- [4] Raspberry Pi Foundation. What Is a Raspberry Pi? [Internet]. Raspberry Pi. 2013. Available from: <https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/>
- [5] What Is Latency? | How to Fix Latency | Cloudflare. Cloudflare [Internet]. Available from: <https://www.cloudflare.com/learning/performance/glossary/what-is-latency/>
- [6] Contributor S. Bandwidth and Throughput in Networking: Guide and Tools - DNSstuff [Internet]. Software Reviews, Opinions, and Tips - DNSstuff. 2019 [cited 2023 May 23]. Available from: <https://www.dnsstuff.com/network-throughput-bandwidth#what-is-throughput-in-networking>
- [7] System throughput (messages per second) [Internet]. www.ibm.com. 2019. Available from: <https://www.ibm.com/docs/en/ztpf/1.1.0.15?topic=performance-system-throughput-messages-per-second>
- [8] Elhadi S, Marzak A, Sael N, Merzouk S. Comparative Study of IoT Protocols [Internet]. Application and Data Analysis for Smart Cities (SADASC'18). Rochester, NY; 2018. Available from: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3186315

- [9] Cui P. Comparison of IoT Application Layer Protocols. Department of Computer Science and Software Engineering at Auburn University [Internet]. 2017 Apr 25; Available from: <https://etd.auburn.edu/handle/10415/5713>
- [10] Evolution of HTTP [Internet]. MDN Web Docs. 2023. Available from: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP
- [11] mozilla. An overview of HTTP [Internet]. MDN Web Docs. 2019. Available from: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>
- [12] HTTP response status codes [Internet]. MDN Web Docs. 2023. Available from: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>
- [13] What is MQTT? Definition and Details [Internet]. www.paessler.com. 2019. Available from: <https://www.paessler.com/it-explained/mqtt>
- [14] Bernstein C, Brush K, Gillis A. What is MQTT and How Does it Work? [Internet]. IoT Agenda. 2021. Available from: <https://www.techtarget.com/iotagenda/definition/MQTT-MQ-Telemetry-Transport>
- [15] The HiveMQ Team. Publish & Subscribe - MQTT Essentials: Part 2 [Internet]. Hivemq.com. 2015. Available from: <https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/>
- [16] XMPP | Instant Messaging [Internet]. xmpp.org. Available from: <https://xmpp.org/uses/instant-messaging/>
- [17] Saint-Andre P. Extensible Messaging and Presence Protocol (XMPP): Core [Internet]. IETF. 2011 [cited 2023 May 23]. Available from: <https://www.rfc-editor.org/rfc/rfc6120#section-8>
- [18] Shelby Z, Hartke K, Bormann C. rfc7252 [Internet]. datatracker.ietf.org. 2014. Available from: <https://datatracker.ietf.org/doc/html/rfc7252>

- [19] Matter Specification Version 1.0 Sponsored by: Connectivity Standards Alliance [Internet]. 2022 [cited 2023 May 23]. Available from: https://csa-iot.org/wp-content/uploads/2022/11/22-27349-001_Matter-1.0-Core-Specification.pdf
- [20] Service Name and Transport Protocol Port Number Registry [Internet]. www.iana.org. 2023 [cited 2023 May 23]. Available from: <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?search=mqtt>
- [21] Steve. MQTT Keep Alive Interval Explained With Examples [Internet]. Steve's Internet Guide. 2017 [cited 2023 May 23]. Available from: <http://www.steves-internet-guide.com/mqtt-keep-alive-by-example/>
- [22] Eclipse Mosquitto [Internet]. Eclipse Mosquitto. 2018. Available from: <https://mosquitto.org/>
- [23] Usage Examples — aiocoap 0.4.7.post0 documentation [Internet]. aiocoap.readthedocs.io. [cited 2023 May 23]. Available from: <https://aiocoap.readthedocs.io/en/latest/examples.html>
- [24] matterapi: A client library for Mattermost API supporting sync/async [Internet]. PyPI. 2022 [cited 2023 May 23]. Available from: <https://pypi.org/project/matterapi/>
- [25] Martin S. pymatter — pymatter 0.1 documentation [Internet]. pymatter.readthedocs.io. 2015 [cited 2023 May 23]. Available from: <https://pymatter.readthedocs.io/en/latest/>
- [26] Matter [Internet]. Google Home Developers. Available from: <https://developers.home.google.com/matter?hl=en>
- [27] Matter [Internet]. GitHub. 2023. Available from: <https://github.com/project-chip/connectedhomeip>

Appendix A: MQTT Implementation Source Code

MQTT Publish Client (Latency)

```
1 import paho.mqtt.client as mqtt
2 import time
3
4 num_messages = 100 # Set this to the number of messages you want to send
5
6 def on_publish(client, userdata, mid):
7     print(f"Message {mid} published successfully")
8
9 client = mqtt.Client()
10 client.on_publish = on_publish
11
12 client.connect("localhost", 1883, 60)
13
14 try:
15     client.loop_start()
16
17     for i in range(num_messages):
18         topic = "test/topic"
19         start_time = time.time()
20         message = f"Hello World, Start time: {start_time}"
21         client.publish(topic, message)
22
23 except KeyboardInterrupt:
24     pass
```

MQTT Publish Client (Throughput)

```

1  import paho.mqtt.client as mqtt
2  import time
3
4  def on_publish(client, userdata, mid):
5      print(f"Message {mid} published successfully")
6
7  client = mqtt.Client()
8
9  client.on_publish = on_publish
10
11 client.connect("localhost", 1883, 60)
12
13 # Publish messages continuously for 10 seconds
14 start_time = time.monotonic()
15 elapsed_time = 0
16 num_messages = 0
17 while elapsed_time < 10:
18     message = f"Hello World, {start_time}"
19     client.publish("test/topic", message)
20     num_messages += 1
21     elapsed_time = time.monotonic() - start_time
22
23 print(f"Published {num_messages} messages in 10 seconds")

```

MQTT Subscribe Client

```

1  import paho.mqtt.client as mqtt
2  import time
3
4  start_time = 0
5  message_bytes = 0
6  received_messages = 0
7
8  def on_connect(client, userdata, flags, rc):
9      if rc == 0:
10         print("Connected to MQTT broker")
11         client.subscribe("test/topic")
12     else:
13         print("Failed to connect, return code:", rc)
14
15  def on_message(client, userdata, msg):
16      global start_time, message_bytes, received_messages
17      message_bytes += len(msg.payload)
18      message = str(msg.payload.decode("utf-8"))
19      start_time = float(message.split(" ")[-1]) # Extract the start time from the message
20      print("Received message:", msg.payload.decode("utf-8"))
21      received_messages += 1
22
23  client = mqtt.Client()
24
25  client.on_connect = on_connect
26  client.on_message = on_message
27
28  client.connect("localhost", 1883, 60)
29
30  client.loop_start()
31
32  start_time_waiting = time.monotonic()
33  while True:
34      if time.monotonic() - start_time_waiting > 10:
35          break
36      pass
37
38  client.loop_stop()
39
40  end_time = time.monotonic()
41  total_time = end_time - start_time_waiting
42  throughput = received_messages / total_time
43
44  print(f"Total time taken to receive {received_messages} messages: {total_time:.2f} seconds")
45  print(f"Throughput: {throughput:.2f} messages per second")
46
47  with open("C:/Users/46729/MQTT/Tester/Throughput.txt", "a") as f:
48      f.write(f"Throughput: {throughput:.2f} messages per second\n")

```

Appendix B: CoAP Implementation Source Code

CoAP Client

```
import logging
import asyncio
import time

from aiocoap import *

logging.basicConfig(level=logging.INFO)

async def main():
    protocol = await Context.create_client_context()

    num_requests = 1000
    total_bytes = 0
    TP_time = time.monotonic()

    for i in range(num_requests):
        request = Message(code=GET, uri='coap://localhost/time')

        start_time = time.time() # Start time

        try:
            response = await protocol.request(request).response
        except Exception as e:
            print('Failed to fetch resource:')
            print(e)
        else:
            end_time = time.time() # End time
            elapsed_time_ms = (end_time - start_time) * 1000
            total_bytes += len(response.payload)

            if(elapsed_time_ms != 0):
                with open("C:/Users/46729//Latency/Latency1000.txt", "a") as f:
                    f.write(f"{elapsed_time_ms}\n")

    TP_endtime = time.monotonic()
    TP_totaltime = TP_endtime - TP_time
    throughput = total_bytes / TP_totaltime
    print(f"Total Requests: {num_requests}")
    print(f"Total Bytes: {total_bytes}")
    print(f"Throughput: {throughput:.2f} bytes/s")

    with open("C:/Users/46729//Scalability/Scalability100.txt", "a") as f:
        f.write(f"Requests: {num_requests} Bytes: {total_bytes} Throughput: {throughput:.2f} bytes/s\n")

if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        pass
    finally:
        asyncio.get_event_loop().close()
```

Appendix C: MATLAB Implementation Source Code

```
x_labels = {'10 seconds', '30 seconds', '60 seconds'};
bar_data = [mean_protocol1; mean_protocol2]';
bar_std = [std_protocol1; std_protocol2]';

figure;
bar_handle = bar(bar_data);
hold on;

num_bars = size(bar_data, 2);
num_groups = size(bar_data, 1);
group_width = min(0.8, num_bars/(num_bars + 1.5));

for k = 1:num_bars
    x_pos = (1:num_groups) - group_width/2 + (2*k - 1) * group_width / (2*num_bars);
    errorbar(x_pos, bar_data(:, k), bar_std(:, k), 'k', 'linestyle', 'none', 'linewidth', 1.5);
end

legend({'CoAP', 'MQTT'}, 'Location', 'northeast');
xlabel('Time frame');
ylabel('Throughput [msg/s]');
title('Throughput Comparison');

bar_handle(1).FaceColor = [0.25098, 0.87843, 0.81569];
bar_handle(2).FaceColor = [0.59608, 0.98431, 0.59608];
grid on;
box on;
hold off;
set(gca, 'XTickLabel', x_labels);

xlim([0.5, num_groups + 1]);
```