

Emil Paulin Andersen

Evaluating Publish/Subscribe Protocols for use in Constrained Networks

Master's thesis in Informatics: Software Engineering

Supervisor: Frank T. Johnsen

Co-supervisor: Michael Engel

June 2022



Norwegian University of
Science and Technology

Emil Paulin Andersen

Evaluating Publish/Subscribe Protocols for use in Constrained Networks

Master's thesis in Informatics: Software Engineering
Supervisor: Frank T. Johnsen
Co-supervisor: Michael Engel
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Table of Contents

1	Introduction	3
1.1	Motivation	3
1.2	Research Objective	4
1.2.1	Research Questions	4
1.3	Research Methodology	4
1.4	Premises	4
1.5	Scope	4
1.6	Contributions	5
1.7	Outline of Thesis	5
2	Background and Previous Research	6
2.1	Application	6
2.1.1	Defense	6
2.1.2	Civilian	7
2.2	Network Models	7
2.3	Publish/Subscribe Pattern	8
2.4	MQTT	9
2.5	MQTT-SN	10
2.6	Quality of Service	10
2.7	Topics	12
2.8	MQTT Packet Format	12
2.8.1	Control Header	12
2.8.2	Remaining Length	13
2.8.3	Variable Header	14
2.8.4	Payload	14
2.8.5	Packet Example	14
2.8.6	Message Sequence Chart	15
2.9	MQTT-SN Packet Format	16
2.9.1	Message Header	16
2.9.2	Message Variable Part	17
2.9.3	Packet Example	17
2.10	ZeroMQ	17
2.10.1	Transport Protocols	18
2.10.2	Socket API	18

2.10.3	ZeroMQ with the Publish/Subscribe Pattern	19
2.10.4	Message Format	20
2.10.5	Intermediary Message Handling	20
2.10.6	Previous Studies on ZeroMQ	21
3	Development	22
3.1	Technology	22
3.2	Supported Protocols	23
3.3	Test Framework	23
3.4	Frontend Application	25
3.5	Data Storage	27
3.6	Complete Architecture	28
4	Test Setup	32
4.1	Netem	32
4.2	MQTT	33
4.3	MQTT-SN	33
4.4	ZeroMQ	34
4.5	Data	36
5	Results	37
5.1	MQTT	37
5.2	MQTT-SN	39
5.3	ZeroMQ with TCP	41
5.4	ZeroMQ with PGM	43
6	Analysis	46
6.1	5G	46
6.2	Tactical Broadband	48
6.3	SATCOM	50
6.4	NATO Narrowband	51
6.5	CNR with 1% Loss	53
6.6	CNR with 10% Loss	55
6.7	MQTT	56
6.8	MQTT-SN	57
6.9	ZeroMQ with TCP	57
6.10	ZeroMQ with PGM	58

6.11 Summary	58
7 Recommendations	60
7.1 Cutoffs	60
7.2 MQTT	61
7.3 MQTT-SN	61
7.4 ZeroMQ with TCP	62
7.5 ZeroMQ with PGM	62
8 Conclusion	64
8.1 MQTT	64
8.2 MQTT-SN	65
8.3 ZeroMQ	65
9 Future Work	66
9.1 Developing an MQTT-SN Client	66
9.2 Additional Testing with ZeroMQ	66
9.3 Configuring the TCP Stack	66
9.4 Open Sourcing the Analysis Tool	66
Appendix	71
A Additional Details About the Analysis Tool	71
B Virtual Machine Configuration	72
C iPerf3	73
D MQTT-SN Setup	74
E ZeroMQ with PGM Setup	75
F Results for MQTT	76
F.1 5G	76
F.2 Tactical Broadband	76
F.3 SATCOM	77
F.4 NATO Narrowband Waveform	78
F.5 CNR with 1% loss	79
F.6 CNR with 10% loss	81
G Results for MQTT-SN	83
G.1 5G	83
G.2 Tactical Broadband	84
G.3 SATCOM	85

G.4	NATO Narrowband	86
G.5	CNR with 1% loss	87
G.6	CNR with 10% loss	88
H	Results for ZeroMQ with TCP	91
H.1	5G	91
H.2	Tactical Broadband	91
H.3	SATCOM	92
H.4	NATO Narrowband	93
H.5	CNR with 1% loss	94
H.6	CNR with 10% loss	95
I	Results for ZeroMQ with PGM	97
I.1	5G	97
I.2	Tactical Broadband	97
I.3	SATCOM	98
I.4	NATO Narrowband	99
I.5	CNR with 1% loss	100
I.6	CNR with 10% loss	100

Acknowledgement

I would like to thank my supervisors, Frank T. Johnsen and Michael Engel, for their excellent support and guidance through this master thesis. They have given me good advice and feedback throughout this semester, always guiding me in the right direction, which I am very grateful for.

I would also like to give a special thanks to my friends and family, who have supported and encouraged me through my years of studying and acquiring my degree. This accomplishment would not have been possible without them.

Abstract

Today, many devices and sensors produce large quantities of data that need to be processed to retrieve valuable information. The number of devices grows as new technology rapidly emerges, which in turn produces even more data that must be handled in one way or another. Many of these devices are simple in design and do not possess the computational power to perform this work themselves. This is especially the case in fields like machine learning, where there is a need for large data sets to be processed in order to identify patterns and provide predictions. Since these devices cannot do this work themselves, there is a significant need for some way to transport this data to another device or system that is capable of handling this task. There are many solutions out there today that are specifically designed to perform this task, and this thesis will investigate several protocols that have this responsibility. The protocols all use the publish/subscribe pattern, an architectural pattern used to exchange messages between systems. The first two protocols we have investigated in this study include MQTT and MQTT for Sensor Networks (MQTT-SN), which are covered by OASIS specifications. MQTT-SN is a simplified UDP-based alternative to MQTT. The third and final protocol we have looked at is ZeroMQ.

To allow systems to exchange messages, there is a need for a network that they can communicate over. These networks are not always reliable and do not always operate in suitable environments that allow for efficient communication. This is especially the case in the defense sector, where they use radio networks with limited bandwidth that is prone to errors in transmission. There are also many scenarios in the civilian sector that requires efficient communications over constrained networks.

Through experimentation, this thesis has evaluated how the MQTT, MQTT-SN, and ZeroMQ protocols have performed in different network environments with different sets of limitations. To conduct these experiments, we have created an analysis tool that is specifically made for this task. The tool was able to run tests with the protocols we wanted to evaluate, and it was also able to emulate the networks we wanted to test the protocols in. Each test provided data on the protocol's performance, which we later analyzed, enabling us to compare the protocols against each other directly. Based on the analysis of each protocol, we have concluded that they are suitable for use in less constrained networks but struggle with efficient communication in the more challenging ones. MQTT-SN stands out among the protocols we have evaluated in that it provides the most efficient and reliable message transportation in most networks.

1 Introduction

During the last couple of decades, there has been a substantial increase in electronic devices used for many different purposes. The devices have become smaller in size, and the computations they have to perform have increased in complexity. Gartner defines the Internet of Things (IoT) as ‘a network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment’ [1]. IoT has become an essential aspect of daily life, helping us automatically manage simple and complex tasks without much hassle. These devices may seem simple on the surface, but as with many computer systems, they depend upon many thoughtful design decisions to develop the software systems. They often have limited resources like storage and computing power to keep costs down. In order to take full advantage of their capabilities, the system needs to be developed around these limitations. The quantity of data produced by these devices is increasing, and it is not always efficient nor possible to process this data on the system locally. Therefore, there has become a need to move the data responsibility to other systems with more computing power and better data processing. The data should be sent from one system to another in an efficient and fault-tolerant manner. This message exchange must happen across a network, which might not always be reliable. This introduces a new problem, which we are investigating in this thesis.

In this thesis, we will investigate several protocols within the publish/subscribe pattern [2] and evaluate their performance in constrained networks. The protocols we will investigate include MQTT [3], MQTT for Sensor Networks (MQTT-SN) [4] and ZeroMQ [5]. The purpose is to compare the protocols and reveal if they are suitable for use when the objective is to transport messages in a manner that can be considered effective. Discovering that a protocol is suitable for use can benefit military use-cases where operations take place in environments where communications can be considered challenging. Not only the defense sector can benefit from this research. Many civilian use-cases exist where communication between systems is essential, for instance, in search and rescue operations, where remote environments can provide many complex challenges for network communication.

Disconnected, intermittent connectivity, and low-bandwidth (DIL) is a definition introduced by Scott et al. [6], which describes key attributes of a network that can lead to issues when systems need to communicate. The term introduces a description of how critical data will need to be delivered in a military scenario, even in the case of DIL networks. The term can also be applicable in other scenarios, especially in the case of natural disasters, where networks have become significantly degraded or replaced by ad hoc networks. To simulate networks that can be considered DIL, we have used several models to introduce limitations on the network with various constraints.

1.1 Motivation

The increase in the use of systems that are responsible for sending large quantities of data has increased the need for ways to achieve this objective in a matter that is both reliable and swift. There has been a significant increase in commercial IoT devices that benefit many users but need to rely on external systems for data aggregation and analysis to provide valuable information. The military faces many scenarios where the need for communication is vital for the commanders and units to achieve their objectives successfully. Operations often occur in locations where the network coverage is constrained and prone to problems. Therefore, they need to rely on networks with low bandwidth and a high packet loss probability. This issue can cause a significant hindrance to the operations due to unreliable means of communication, which can be critical to mission success.

1.2 Research Objective

This research aims to evaluate several publish/subscribe protocols and investigate their performance in networks that are limited due to circumstances in the environment where they are deployed, specifically networks that are considered to have DIL characteristics. The metrics we have investigated are from both the network and application layers. On the network layer, we specifically want to look at the total number of packets sent, data rate, message sizes, retransmissions, and duplicate acknowledgments. For the application layer, we want to look at the number of messages sent, the packet loss, transmission delay, and the number of times the clients have disconnected during the testing phase. The most important metrics we are looking into are the transmit delay and packet loss on the application layer. The reason is that in the use cases we are investigating, the protocols require a high degree of efficiency. Therefore, the metrics gathered on the network layer will be used as additional information to supplement and give reason to our main findings.

1.2.1 Research Questions

We have identified two research questions that will answer our research objective for this thesis.

1. **RQ1:** Can we correctly evaluate the performance of publish/subscribe protocols in DIL networks?
2. **RQ2:** Which DIL networks are the publish/subscribe protocols most suited for?

1.3 Research Methodology

This thesis uses an empirical research method described in Wohlin et al. [7]. The research bases itself on controlled experimentation, a quantitative research method. Experimentation focuses on observing changes in a highly controlled setting, manipulating some variables while others are fixed to measure changes. This requirement is fulfilled by the test environment we have in our analysis tool. The fixed variables in our experiments have been the protocol settings which are not changed during a testing session. The variables we are manipulating are the settings used to emulate the various DIL networks. The data collected then reveal the key factors that determine the experiment's outcome. There is a focus on validity when performing experiments to ensure that the results are not affected by external variables.

1.4 Premises

This research is a continuation of previous work conducted during the preparatory project (IT3915) for the master's thesis [8]. We developed an analysis tool for testing and analyzing publish/subscribe protocols during that project, which we continue to use during this thesis. Therefore, we have data from testing using the MQTT protocol which we are including in this study in order to compare and evaluate it to two other protocols, namely MQTT-SN and ZeroMQ. We are also implementing new features in the analysis tool to support these protocols.

1.5 Scope

The work in this thesis continues the previously conducted research on the evaluation of publish/subscribe protocols in distributed systems. NATO IST-150 (NATO Core Services profiling for Hybrid Tactical Networks) is a research task group that has the

objective of investigating Service-Oriented Architecture (SOA) in the tactical domain [9]. The group has analyzed standards and techniques that can be suitable for use in networks characterized as DIL [10]. They have experimented with the MQTT protocol for use in tactical networks, and their findings show that the protocol is promising for these use cases. ZeroMQ also finds itself to be one of the more promising protocols for use in constrained networks [11].

For this reason, we further investigate these protocols to determine their usability for the use-cases mentioned. Other studies in the context of the IST-150 group has also evaluated the performance of both the MQTT and MQTT-SN protocol [12]. The study found that they were suitable alternatives to using protocols like CoAP [13], which is purposefully made to be used in constrained networks [14]. Therefore, the goal of this research is to further the studies previously conducted by the research task group within the scope of publish/subscribe protocols.

This thesis focuses on evaluating the protocols based on the ‘L’ in DIL, meaning that the networks we use have considerably low bandwidth. The reason for not investigating the ‘D’ and ‘I’ is that the protocols we are using do not function without access to a network, and there is no purpose for us to test these protocols in this manner.

Several security features are built into the protocols we have looked at and are optional parameters that can be used if needed. We have decided not to focus on the security aspects related to distributed communication. Security is an important aspect but requires many considerations, which due to time constraints, we have not pursued during this study.

1.6 Contributions

The contributions of this thesis are twofold: First, we have developed a test environment for performing repeatable, controlled, and reliable protocol tests and evaluations. Second, this study contributes by extending the knowledge of publish/subscribe protocols and their use in scenarios where reliable and quick communication is vital to operations. The comparisons between the protocols make their strengths and weaknesses clear for different situations. The recommendations can help select the appropriate options when implementing a solution where these requirements are of high importance.

1.7 Outline of Thesis

The thesis is organized as follows: Chapter 2 describes the background for what we have investigated as well as details about each of the protocols we have looked at. Chapter 3 describes the development and structure of the analysis tool we have developed. Chapter 4 gives an overview of the test setup we used when conducting tests for each of the protocols and the configurations we have used. Chapter 5 provides the combined results for each test we have conducted. The complete description from each test can be found in appendices F to I. Chapter 6 provides a detailed description of the analysis we performed. In chapter 7 we give recommendations for the use of the protocols in each of the networks. Chapter 8 provides our conclusion to the research done in the thesis and provides answers to our research questions. In chapter 9 we have recommendations for what to look into in the future. The appendices from A to E gives a detail description the setup we have used, while appendices F to I as mentioned gives the a full description of each result we had during testing.

2 Background and Previous Research

This section describes the reasons we have to investigate our objective in this research. Based on the current needs and challenges faced, the use cases and applications for the defense and civilian sectors have been described. Next, we will describe the network models used to emulate the DIL networks. We also detail the publish/subscribe pattern used by the protocols described in this chapter. In the preparatory project for this thesis, we have previously provided the details on the MQTT and MQTT-SN protocols. The sections related to Quality of Service (see section 2.6), Topics (see section 2.7) and Packet Format (see section 2.8 and 2.9) have been taken from this project, including the figures and tables.

2.1 Application

2.1.1 Defense

An important criterion when military forces are operating at the tactical level is situational awareness, which requires a need for regular communication and rapid information sharing. In military operations, there is a need for units and commanders to communicate about evolving situations using devices with message-sharing capabilities over networks that often can be inconsistent and lacking in bandwidth. Therefore there is a demand for ways to ensure that this data exchange transpires rapidly, without loss of information. Some cases also require a large volume of data due to the need to share vital information during the operation. This data can range from low-frequency data points like GPS coordinates of units and vehicles to video recording and streaming in real-time.

Combined operations is a concept used by the military which describes operations that rely on tight cooperation by different units of the armed forces in order to achieve the mission objective [15]. This concept involves soldiers on the ground and support services like air and naval forces, communicating and sharing intelligence through an information network. Research and development within the defense sector have produced increasingly advanced technology and hardware that will support the modern soldier in a digitally connected environment.



Figure 1: Movement data of the mechanized battalion from the Anglova scenario [16]

The act of research and experimentation is not always practical to do out in the field and is often relegated to experimentation in labs. During these experiments, the issue arises

that the results might not always apply to the real world. This concern is especially the case in the military, where the variables found in real-world scenarios are inherently difficult to account for. This problem can be solved either through simulation or emulation, which can provide more accurate representations of conditions faced during a real-world situation. The Anglova scenario is an emulation scenario developed as part of the NATO IST-124 Research Group and is used for experimentation over tactical network environments similar to the ones we have investigated [17]. The scenario simulates a realistic military operation with a mechanized battalion and provides positional movement data, which can be seen in figure 1. Anglova has been used by researchers in order to experiment and test protocols and is therefore proven as a good scenario that provides a realistic military setting for experimentation.

Examining and comparing the performance of protocols specialized for this purpose can provide meaningful recommendations for use cases that provide better and more efficient communication for military forces.

2.1.2 Civilian

During emergencies, like natural disasters, there is a need for operations to be conducted to save lives. Most of these operations rely on communications between people with different duties and responsibilities, which implies that different systems need to be able to talk to each other. There is also a need for victims of a disaster to communicate with rescue workers or broadcast their position. Many emergencies happen in locations outside the area covered by networks that provide stable connections, which can impose a problem for data transmission. There might still be issues with communications in locations with developed infrastructure, as the infrastructure is likely to be damaged due to the disaster. The volume of data that needs to be transmitted can also cause a hindrance by bottlenecking certain critical systems. The need for reliable and fast communication is therefore essential in these scenarios.

There has been several advancements made within IoT and Smart Cities technologies in later years, which means that a lot of sensors and data can be available in the case of a disaster situation [18]. The concept of Smart Cities is relatively new. It can be defined as a place where networks and services can be used more efficiently by the use of new technology that provides benefits in managing the urban environment [19]. Disaster situations might also occur in Smart Cities, and there is a need to plan for such events by using the assets these cities provide. Agencies responding to the events will need to quickly connect to the available resources and systems that can provide them with information about many vital aspects that can supply an overview of the situation and help save lives quickly.

By understanding how the protocols behave and how their performance is affected when deployed in networks that simulate these environments, we can better understand how to implement them into systems.

2.2 Network Models

In order to simulate DIL networks with varying degrees of constraints, we have designed several network models that emulate realistic networks used by the military and NATO today, as mentioned in section 2.1.1.

The network models can be seen in table 1. We have used the same network models that we have previously used in the preparatory project [8]. However, We have added a model to simulate low-band 5G networks, as this is an area of interest for military powers around the world, where many are looking at integrating 5G technology into their military networks [20].

Tactical Broadband radios are designed to be used in challenging outdoor environments, i.e., the tactical battlefield [21]. The radios will provide secure and reliable communication capabilities for outdoor use, where the communication distance can be up to 100 kilometers during line of sight conditions. This network model provides relatively high throughput and reliability compared to the other networks we have used.

SATCOM is a satellite communications service that NATO has used for over 15 years. The service provides super high frequency (SHF) and ultra-high frequency (UHF) communications access. SHF is used for static and deployed stations on the ground, while UHF is used for tactical communications [22]. In this thesis, we have focused on testing with a SATCOM model more similar to what is provided by UHF.

NATO Narrowband Waveform is a standard developed by NATO to provide a single-channel ad hoc network that will serve voice and data traffic [23].

Combat Network Radio (CNR) is a type of radio that operates in a push-to-talk manner, with the purpose of command and control of combat, combat support, and combat service support during military operations [24]. It operates in a network that provides both a half-duplex circuit and either a single radio frequency or a set number of radio frequencies.

The defense sector uses the networks covered in this section, but there are also cases where these networks are shared and used by the civilian sector. Humanitarian Assistance and Disaster Relief (HADR) is an example of where the civilian meets the military, requiring interoperability, and where sharing of networks and data becomes important [25]. Therefore, the network models are relevant for not only defense but also for civilians.

Network	Data rate	Latency	Loss percentage
Low-band 5G	100 mbit/s	20 ms	0%
Tactical Broadband	2 mbit/s	100 ms	1%
SATCOM	250 kbit/s	550 ms	0%
NATO Narrowband Waveform	16 kbit/s	500 ms	0%
CNR	9.6 kbit/s	100 ms	1% / 10%

Table 1: Tactical network models

2.3 Publish/Subscribe Pattern

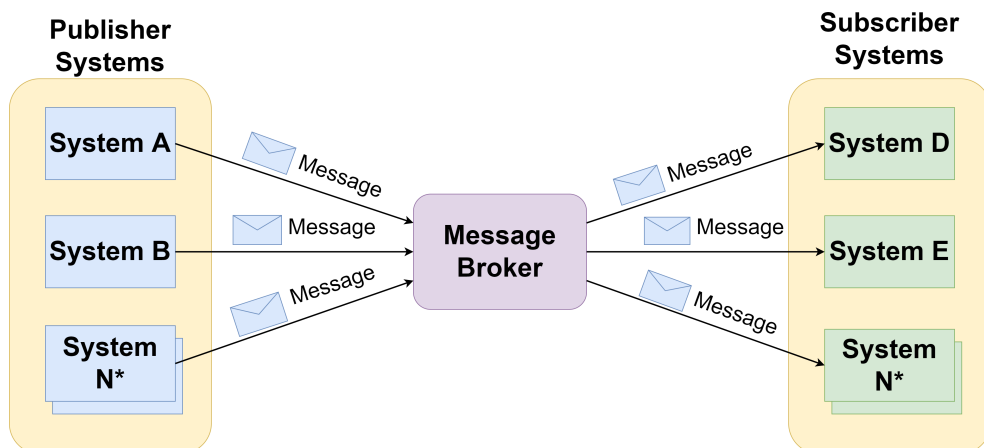


Figure 2: Publish/Subscribe pattern

The publish/subscribe messaging pattern is used for communication between systems where some have the role of producing information that will be exchanged with another

that subscribes to that information. This research compares the MQTT, MQTT-SN, and ZeroMQ protocols, which all fall inside this communication paradigm.

The publish/subscribe pattern requires that there are producers and consumers. MQTT and MQTT-SN also require an intermediary that can handle the filtering and distribution of messages from publisher to subscriber. Figure 2 shows an overview of this architecture with both publishers and subscriber systems, as well as an intermediary message broker. There can be an arbitrary amount of publishers or subscribers, and they do not have to correspond exactly. One could, for example, have one publisher and ten subscribers, or vice versa. MQTT uses a broker as an intermediary responsible for managing the connections with the publisher and subscriber systems. With this implementation, the clients do not need to be aware of each other; they are only required to know about the message broker to forward or receive messages. The initial connection setup requires that the client forwards some information about themselves to the broker. The setup configuration can deviate between implementations, but the broker usually needs information about the client's name, role, and protocol-specific details. When a publishing client forwards a message to the broker, the broker needs to handle filtering to send the message to the correct clients. This process happens through a topic-based system in MQTT, but other protocols may also use content-based systems. With the topic-based system, the publisher clients decide a topic for the message it wants to send, and the subscriber clients that wish to receive this message subscribe to the broker with the same topic. This allows the broker to filter out the clients that have not subscribed and send the message to the clients that have.

2.4 MQTT

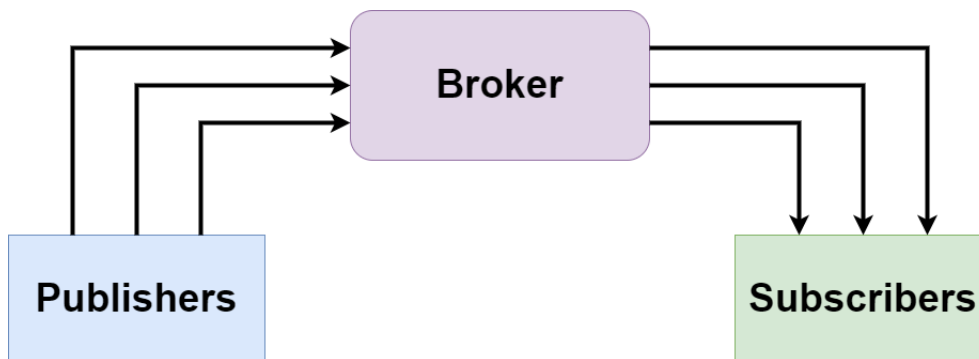


Figure 3: General MQTT architecture

MQTT is an industry-standard protocol for message transfer that allows for different levels of Quality of Service (QoS). MQTT uses the publish/subscribe pattern and therefore uses publishers, subscribers, and a broker, as can be seen in figure 3. Here the messages are fed from the publisher into the broker, which sends the messages to the subscriber. The protocol is lightweight, provides low overhead, and is suitable in environments that have challenging network topologies. It provides benefits in persistent sessions and a high level of scalability. MQTT usually uses TCP for data transmissions, but this is not required. The only requirements for implementing the protocol are that the transport protocol is ordered, provides bi-directional connections, and has lossless communication. This means that, in theory, MQTT could be implemented using another protocol for transport, e.g., Stream Control Transmission Protocol (SCTP) [26]. The use of TCP provides additional guarantees in combination with the QoS settings but increases the overhead for the protocol to some extent. MQTT is also data-agnostic, meaning that it allows the use of many data formats in its messages. The size requirements for MQTT limit the payload of a message to approximately 256 MB [27].

2.5 MQTT-SN

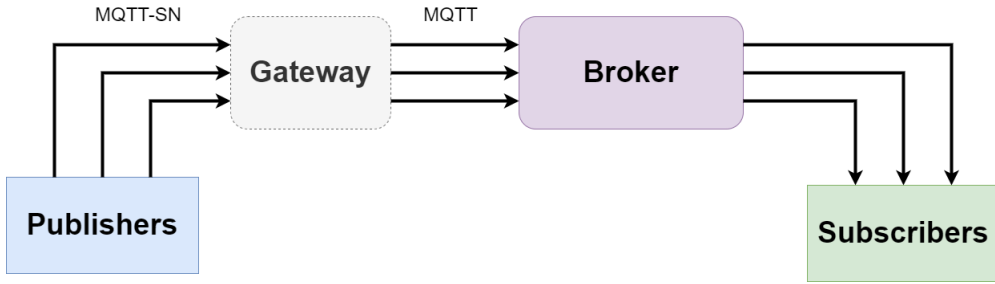


Figure 4: General MQTT-SN architecture

The MQTT-SN protocol is an alternative to the MQTT protocol while sharing many of its characteristics. Like MQTT, the MQTT-SN protocol has both topics and QoS settings. The main difference between the protocols is that, unlike MQTT, MQTT-SN does not require a reliable transport protocol, thus removing the need for a permanent connection. The transport protocol used by MQTT-SN is UDP. Using UDP means that the protocol does not provide the same insurance for message delivery as MQTT, which uses TCP. The use cases for MQTT-SN are mostly in applications that run on low power and low-cost devices, like sensors. These devices are usually basic in their complexity and send out small amounts of data with varying frequencies. MQTT-SN has an advantage over MQTT in that it reduces the size of the message by changing how it handles topics. The topic implementation is detailed in section 2.7. The protocol also contrasts with MQTT in the use of gateways. A gateway is responsible for receiving MQTT-SN messages and converting them into MQTT messages that are forwarded to a broker. There are two types of gateways in MQTT-SN: the transparent gateway and the aggregate gateway. A transparent gateway receives one or more MQTT-SN connections from clients and converts these into the same number of MQTT streams to the broker. An aggregate gateway also takes one or more MQTT-SN streams but instead only produces a single MQTT stream to the broker. Figure 4 shows the general architecture of MQTT-SN using a transparent gateway, as can be seen by the matching number of MQTT-SN and MQTT connections to and from the gateway.

2.6 Quality of Service

Both MQTT and MQTT-SN provide different levels of quality of service. There are 0 (fire and forget), 1 (at least once), and 2 (only once). Publisher clients set the QoS level when publishing messages, sending it with each published message. The subscriber clients specify the QoS they wish to have by sending a subscription message to the broker containing the QoS level. It is important to note that if a message is published with a higher QoS than what the subscriber has subscribed to, the message will be sent from the broker to the subscriber using the QoS defined in the subscription, not what the publisher has set when sending the message. This means that if the subscriber has set QoS to 0, there is no guarantee that the message arrives in the end even if it was published using QoS 1 or 2. MQTT-SN also has an additional level, 3 or -1, which provides additional services not found in MQTT ¹. On the publisher side, they decide the QoS when publishing a message. This study will be limited to QoS 0 and 1 for MQTT-SN, while tests from MQTT will use all QoS levels.

¹The setting is known as -1, but the flag used in messages is set to the decimal number 3. This setting reduces some of the requirements related to client connections and must use one of two specific topic types [28].

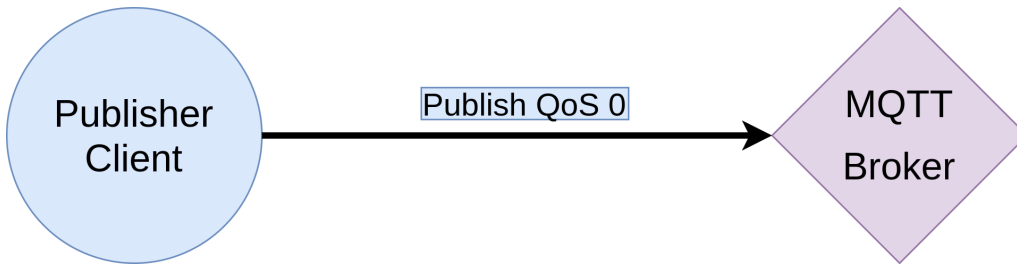


Figure 5: Packet sent using QoS level 0

Table 5 shows a message being sent using QoS 0. The publisher client is shown sending a single packet containing the message to the broker. The packet is not guaranteed to be delivered and, therefore, should be used on low-priority packets.

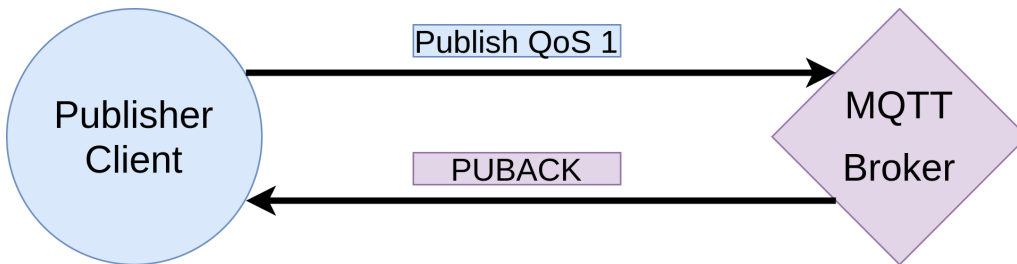


Figure 6: Packets sent using QoS level 1

Table 6 shows a message being sent using QoS 1. The publisher sends a packet with the message to the broker, which returns an acknowledgment packet confirming that the message was received. It is important to note that this level of service does not ensure that duplicate messages do not arrive at the other end. This setting is used for messages of some importance and must arrive, but where duplicates do not matter.

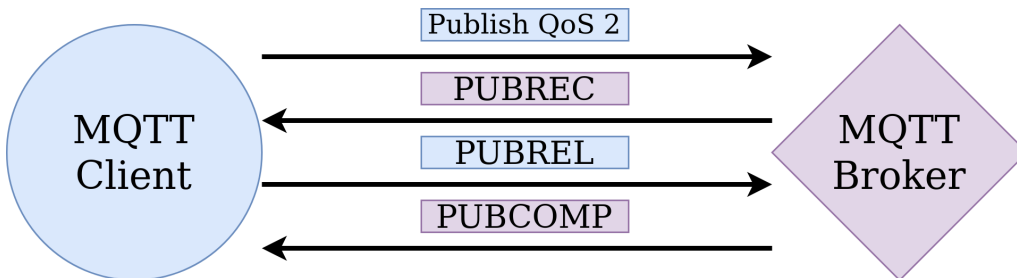


Figure 7: Packets sent using QoS level 2

Table 7 shows a message being sent using QoS 2. The publisher sends a message to the broker and receives a packet acknowledging the message. If the publisher does not receive this message from the broker, it sends a new packet containing the same message. In addition to the message, it also sets the duplicate flag in the packet to let the broker know that this is not a new message. When the publisher retrieves an acknowledgment, it stores the packet and discards the original message. At this time, the publisher sends a new packet which, in return, the broker responds with a comply packet, ending the transmission. This solution does cause significantly more overhead than the other QoS levels but does ensure that no duplicate messages arrive.

2.7 Topics

Topics are used by both the MQTT and the MQTT-SN protocol and are a way for clients to share information. The format of the topics is UTF-8 strings with words that give meaning to a category or data source, delimited by a forward slash '/'. A typical example of a topic is, 'MyHouse/Basement/Temperature', which implies that the topic is temperature data for the basement in your house. The publishing client is responsible for registering the topic, which it will send relevant data about to the broker. The subscriber is then responsible for subscribing to the corresponding topic. A subscribing client is not only limited to a single topic and can subscribe to multiple topics by the use of wildcards [29].

There are some differences in how MQTT-SN handles topics related to MQTT. In MQTT-SN, the topic name is replaced by a short, two-byte topic id. This change aims to reduce the strain on bandwidth as data is not published with the full topic name, as is the case in MQTT. Instead, a topic name is registered through a procedure where clients register a topic name with the gateway. The gateway will assign a topic id and return it to the relevant clients if the registration is accepted. The publishing client will then use this topic id when publishing messages. There is an option where we can use pre-defined topics, meaning there is no need for a registration procedure. In this case, the topic id has already been mapped to a topic name that is known by both the clients and gateway. This is indicated by a flag set in the message. The subscriber still needs to subscribe to a pre-defined topic id. Lastly, there is an option to use short topic names. These names have to be a fixed length of two octets. Like with a pre-defined topic id, there is no need to register the topic in this case. Both pre-defined and short topics still require the client to subscribe using the correct topic id.

2.8 MQTT Packet Format

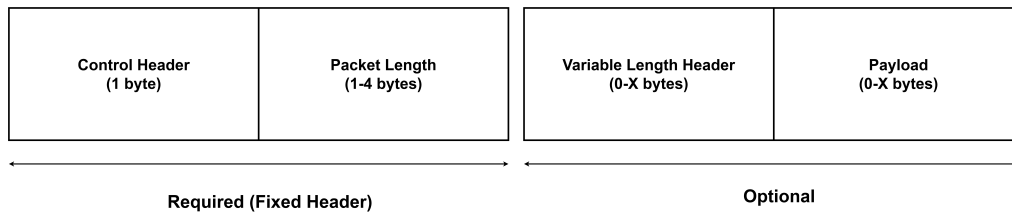


Figure 8: MQTT Message packet format

Figure 8 shows the packet format of MQTT messages. The packet header consists of a single byte control field and a remaining length field that varies from 1 to 4 bytes. These two fields are the fixed header and must be included in every message. This means that the smallest size of any MQTT packet will be two bytes long, as these fields are required. In addition to the fixed header, there is also the variable header and the payload, which can be of varying lengths and only present in some packets.

2.8.1 Control Header

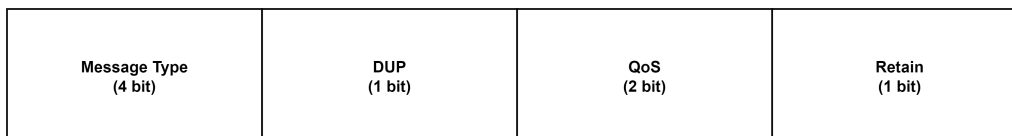


Figure 9: Control header

The control header contains two fields, consisting of 4 bits each. The first field represents the command type of the packet. For example, a CONNECT message will contain the value of 1, telling the receiving broker that this is a connect message. The remaining 4 bits of the control header are reserved for control flags. For a publish command to the broker, the 0th bit tells the broker if the message will be retained, the 1st and 2nd bit tells the broker what QoS level the message is, and the 3rd bit tells the broker whether the message is a duplicate or not.

Table 2 shows an overview of the message types used in the MQTT protocol. There are a total of 16 message types. The value column shows the value that the message type represents in the first four bits of the control header. The direction of flow shows who can send and receive the packet. CONNACK messages can only be sent from the server, meaning the message broker. A message like PUBACK is only used with QoS 1, while PUBREC, PUBREL, and PUBCOMP are used with QoS 2, which can be seen in figure 6 and 7.

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

Table 2: Control packet types from the MQTT version 3.1.1 specification [3]

2.8.2 Remaining Length

This field is reserved for information about the length of the variable header and the size of the payload in the message. The 7th bit in each byte is reserved for a continuation flag. The continuation flag is needed to know if the remaining length field continues into another byte and is part of the remaining length field.

2.8.3 Variable Header

The variable header contains the packet identifier. The information found in the identifier is the protocol name, the protocol level, and the connection flags. Before the protocol name, there is a two-byte long field representing the protocol name's length. These connection flags can be seen in table 3. Will is used if a client disconnects to let other clients know that the original client is currently offline. This is only utilized in the case that a clients disconnect abruptly. How the will is handled can be set by the flags. The packet identifier is not required for every packet. Examples of packets requiring a packet identifier are PUBLISH, PUBACK, SUBSCRIBE and UNSUBSCRIBE.

Bits	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS	Will Flag	Clean Session	Reserved	

Table 3: Connect flags in variable header

2.8.4 Payload

The payload contains the data that is sent with the message. This field is optional as some messages do not need the sending of data. A DISCONNECT message is an example of a packet that does not have a payload.

2.8.5 Packet Example

Table 4 shows an example of what an MQTT packet looks like. The message is of type CONNECT, which can be seen from the value of the fixed header. None of the control flags are set in this example. The remaining length is 17 bytes, including the variable header and payload. Next comes the variable header, which first provides the length of the rest of the protocol name. Then comes the protocol name and version before the connection flags. In this case, the connection flag is set for a clean session. The keep alive for this client is set to 60 seconds. Lastly, there is the payload, where the first two bytes provide the length, and the rest is the Client id. In this case, the client id is 'hello'.

Byte	Field	Hex	Value
1	Fixed Header	0x1	1
2	Remaining Length	0x11	17
3	Variable Header Length	0x0	
4		0x4	4
5	Protocol and Version	0x4d	M
6		0x51	Q
7		0x54	T
8		0x54	T
9		0x4	4
10	Connection Flags	0x2	2 (Clean Session)
11	Keep Alive	0x0	
12		0x3c	60
13	Payload Length	0x0	
14		0x5	5
15	Payload Data	0x68	H
16		0x65	E
17		0x6c	L
18		0x6c	L
19		0x6f	O

Table 4: Example of a packet containing a CONNECT message

2.8.6 Message Sequence Chart

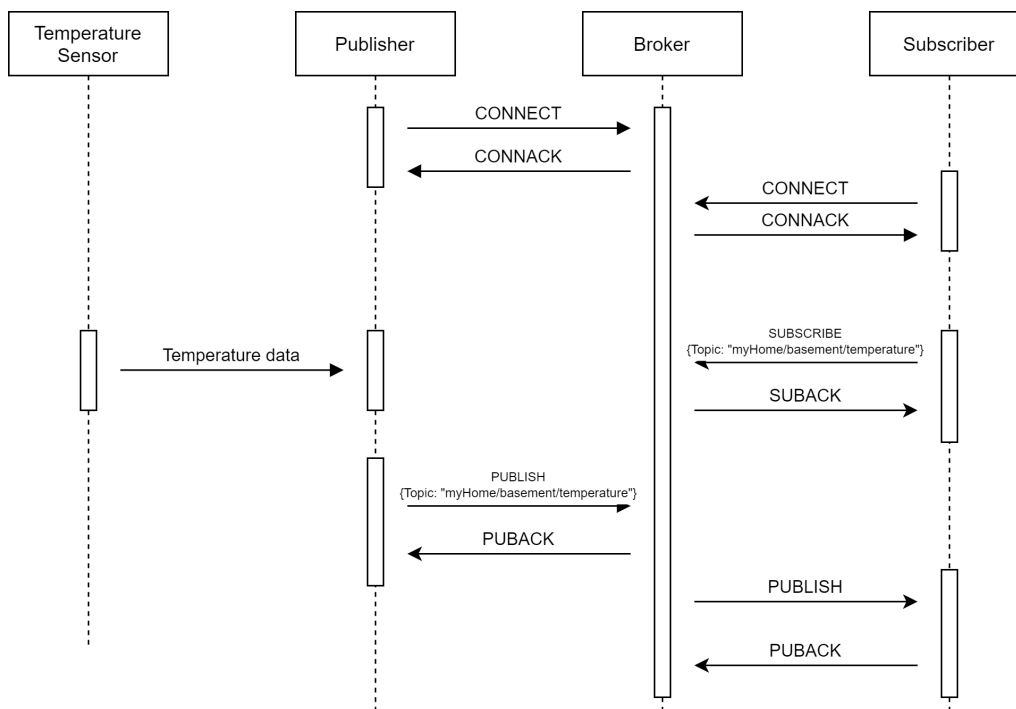


Figure 10: Message sequence diagram

Figure 10 shows a message sequence chart of a typical message transaction using MQTT with QoS level 1. Here, both clients connect to the broker and receive a reply that the connection is successful. Once the subscribing client has connected, the publisher can begin publishing messages. Since we use QoS 1, the broker responds with a PUBACK message to acknowledge that the packet has been received. When the PUBACK message has been sent, the broker can send the message to the subscriber. Since the subscriber

has subscribed with QoS 1, they respond with a PUBACK message that lets the broker know that they have received the message.

2.9 MQTT-SN Packet Format

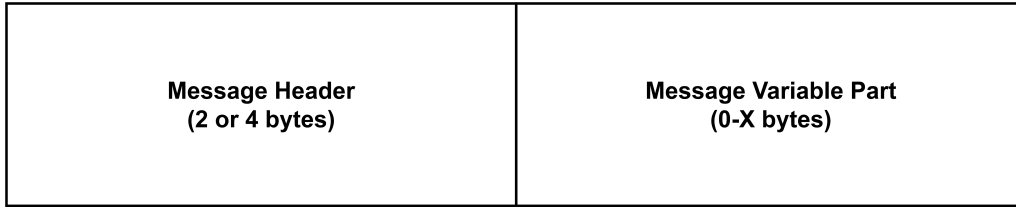


Figure 11: MQTT-SN message packet format

The MQTT-SN packet format is slightly different from the MQTT format, as can be seen in figure 11. It consists of a message header and a message variable part. The message header is either two or four bytes long and is required in every MQTT-SN message. The message variable part is not required for all messages.

2.9.1 Message Header

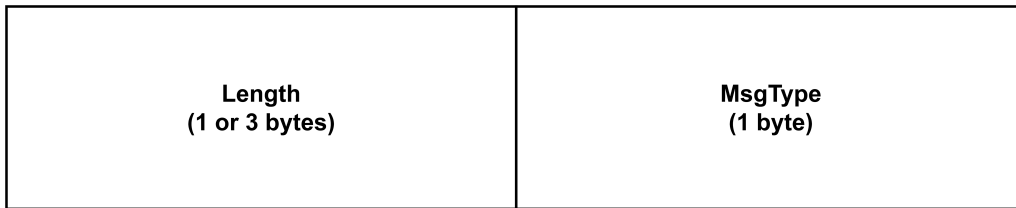


Figure 12: Message header

Figure 12 show the format of the message header. The length field indicates the length of the of the message. If the first byte has the value 0x01, then the length field is three bytes long, where the remaining two bytes specify the total size of the message. This means that the message has a maximum length of 65 535 bytes. If the message has a size smaller than 256 bytes, the first byte is used to denote the length. MQTT-SN does not support message fragmentation.

The message type field is one byte long and is reserved for the message type. Table 5 show the message types in MQTT-SN and their value.

MsgType Field Value	MsgType	MsgType Field Value	MsgType
0x00	ADVERTISE	0x01	SEARCHGW
0x02	GWINFO	0x03	reserved
0x04	CONNECT	0x05	CONNACK
0x06	WILLTOPICREQ	0x07	WILLTOPIC
0x08	WILLMSGREQ	0x09	WILLMSG
0x0A	REGISTER	0x0B	REGACK
0x0C	PUBLISH	0x0D	PUBACK
0x0E	PUBCOMP	0x0F	PUBREC
0x10	PUBREL	0x11	reserved
0x12	SUBSCRIBE	0x13	SUBACK
0x14	UNSUBSCRIBE	0x15	UNSUBACK
0x16	PINGREQ	0x17	PINGRESP
0x18	DISCONNECT	0x19	reserved
0x1A	WILLTOPICUPD	0x1B	WILLTOPICRESP
0x1C	WILLMSGUPD	0x1D	WILLMSGRESP
0x1E-0xFD	reserved	0xFE	Encapsulated message
0xFF	reserved		

Table 5: MQTT-SN message types from the MQTT-SN specification [4]

2.9.2 Message Variable Part

The message variable part contains information that depends on the message type. Relevant information in the variable part can be client id, data, flags, duration, topic id, topic name, etc.

2.9.3 Packet Example

Table 6 shows an example of a CONNECT message sent with MQTT-SN. The message is similar to the CONNECT message sent with MQTT, but the total size of the message is reduced.

Byte	Field	Hex	Value
1	Length	0xB	11
2	MsgType	0x04	4
3	Flags	0x2	2 (Clean Session)
4	ProtocolId	0x1	1
5	Duration	0x0	
6		0x3c	60
7	ClientId	0x48	H
8		0x45	E
9		0x4c	L
10		0x4c	L
11		0x4f	O

Table 6: Example of a packet containing a CONNECT message

2.10 ZeroMQ

ZeroMQ (also called ØMQ, 0MQ, or zmq) is a messaging library that provides asynchronous transactions between distributed or concurrent applications. Unlike the other protocols we have investigated in this study, ZeroMQ does not require an intermediary message broker, which explains its naming. Removing an intermediary can provide benefits in that there will not be a single point of failure. The protocol can use a variety

of patterns, e.g., request-reply, publish-subscribe, and client-server. We have used the publish/subscribe pattern since that is the focus of our thesis. The developers state two goals that explain their reasoning for developing the library: to solve the issues many developers face when needing to connect any code to any other code, anywhere, and the second is making it as easy as possible to use [30]. It is lightweight and fast, similar to the characteristics found in MQTT and MQTT-SN, but it is also advertised as being more configurable.

2.10.1 Transport Protocols

ZeroMQ supports different methods of transportation, including In-Process (INPROC), Inter-Process (IPC), Pragmatic General Multicast (PGM) [31] and TCP [32]. INPROC and IPC are mainly used for local communication transportation within an application running the same process. PGM is a reliable multicast protocol that detects packet loss or out-of-order messages, unlike UDP. Since we have used TCP as a transport protocol during our testing with MQTT, we also chose to use this for ZeroMQ to make the comparisons more in line with each other. In addition to testing with TCP, we have also tested using the PGM protocol, the only multicasting protocol we have used during testing. All other protocols have used unicast. This allows us to see if there are any benefits to transporting messages in this way, and we can compare reliable unicast and reliable multicast in ZeroMQ.

PGM shares similarities to both UDP and TCP. The similarity with UDP is that both protocols support multicasting, where one destination can address data to many other clients. It also shares the characteristics of TCP by the fact that it is aimed at applications that require ordered and duplicate free data transmission and require good reliability [33].

2.10.2 Socket API

ZeroMQ provides a socket API that provides an easy-to-use interface that hides much of the more complex message processing underneath. In ZeroMQ, sockets can be used to initialize, close, configure, send, and receive messages. Sockets can either bind or connect to each other. According to the documentation, the server should use the binding method while the client should use the connect method [34]. Some of the changes to the regular sockets include: one socket can have many outgoing and incoming connections, the connections happen in the background and if a connection is broken it will try to re-establish it, and the connections can not be managed by the application directly, this happens through the socket API. Unlike TCP sockets that stream bytes, ZeroMQ sockets carry messages designed for performance.

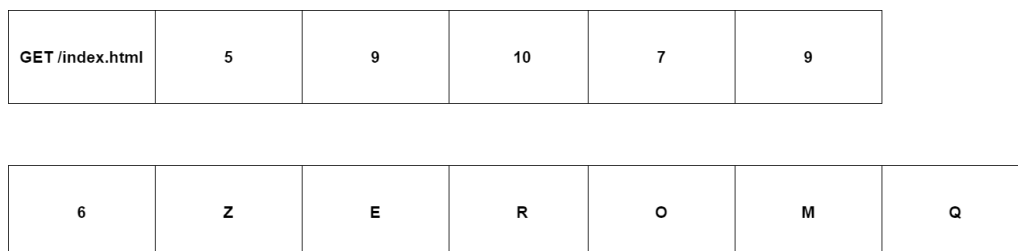


Figure 13: Difference between HTTP and ZeroMQ on the wire

ZeroMQ is not a net neutral carrier, meaning that it is not compatible if one tries to send HTTP requests or receive HTTP responses. This compatibility issue is due to the difference in framing between HTTP and ZeroMQ messages, shown in figure 13.

Sockets have different types depending on the messaging pattern utilized. The type used

will determine the semantics of the socket and how it will manage routing and message queuing. Table 7 shows an overview of the different socket types based on the messaging pattern. In this thesis, we have focused on the publish/subscribe pattern consisting of the PUB, XPUB, SUB, and XSUB sockets. The PUB socket type distributes messages to all connected clients and works unidirectional. The SUB socket is used by a subscriber to subscribe to messages coming from a publisher client. The PUB socket is not allowed to receive messages, while the SUB socket is not allowed to send messages. The XPUB socket works similarly to the PUB socket but allows the reception of subscribe messages. The same is true for the XSUB socket, in that it can send subscribe messages. Their practicality is found when implementing an intermediary between many publishers and subscribers. The intermediary will read subscriptions from the XSUB socket and send them to the XPUB socket. The benefit of using XPUB and XSUB sockets is that we get filtering through the intermediary, similar to a message broker.

Message Pattern	Socket Name	Short Description
Request-Reply	REQ	Used by a client to send requests to and receive replies from a service
	REP	Used by a service to get requests and return replies to a client
	DEALER	Asynchronous replacement for REQ
	ROUTER	Asynchronous replacement for RES
Publish/Subscribe	PUB	Publishes data from a publisher client
	XPUB	Can receive subscriptions from clients in the form of receiving messages
	SUB	Subscribes to data being sent from a publishing client
	XSUB	Can send messages with a subscription to another socket
Pipeline	PUSH	Can talk to anonymous PULL clients and send messages
	PULL	Can talk to anonymous PUSH clients and receive messages
Exclusive Pair	PAIR	Can connect to a single client and send or receive messages. Does not perform filtering
Client-Server	CLIENT	Can talk to one or more SERVERS and send or receive messages
	SERVER	Can talk to zero or more CLIENTS and send or receive messages

Table 7: Socket types and their respective pattern in ZeroMQ

2.10.3 ZeroMQ with the Publish/Subscribe Pattern

ZeroMQ works in the same way as MQTT and MQTT-SN in that a subscription must be set before a subscribing client can receive any messages. The subscriber can subscribe to a number of messages that can also be unsubscribed at any time. The PUB and SUB socket pairs are asynchronous and can be started independently of one another. Since they are asynchronous, there might be a situation where the publisher publishes some messages that the subscriber does not receive. This will often happen because of the time it takes a client to initialize a TCP connection, and this requires some synchronization between the two client types.

The subscriber can connect to more than one publisher. The protocol will try to interleave the incoming messages so that no publisher causes bottlenecking issues. The publisher requires at least one subscriber to send messages. If no subscribers are connected, it will drop all messages. In some cases, the subscriber is slower than the publisher. In this case, the publisher will queue up messages sent to the subscriber in the future. This can cause

some issues but can be handled through the use of high water marks. High water mark is one of the options available to tune with ZeroMQ and allows us to set a number on the messages that will be stored in the queue [35]. If the limit is reached on the socket, it will enter what is called an exceptional state depending on the socket. It will then either block or drop the messages. In version 3 or higher, the filtering of messages happens on the publisher when using TCP or IPC. When using PGM, the filtering happens on the subscriber side.

2.10.4 Message Format

In ZeroMQ, a message can either be one or more parts. Each message part is a `zmq_msg_t` object, and the parts can either be sent separately in the low-level API or as a wrapped multipart message in the higher-level API. Messages can be zero-length which can be used for signaling between systems. A message that is in the process of being sent will not be transmitted immediately, meaning that it will need to fit into memory in the case of larger multipart messages. A multipart message is not sent on the wire until the last part is sent. This also means that we will either receive all parts of a message or none. While ZeroMQ runs on top of TCP, it can provide higher throughput from this message batching technique.

2.10.5 Intermediary Message Handling

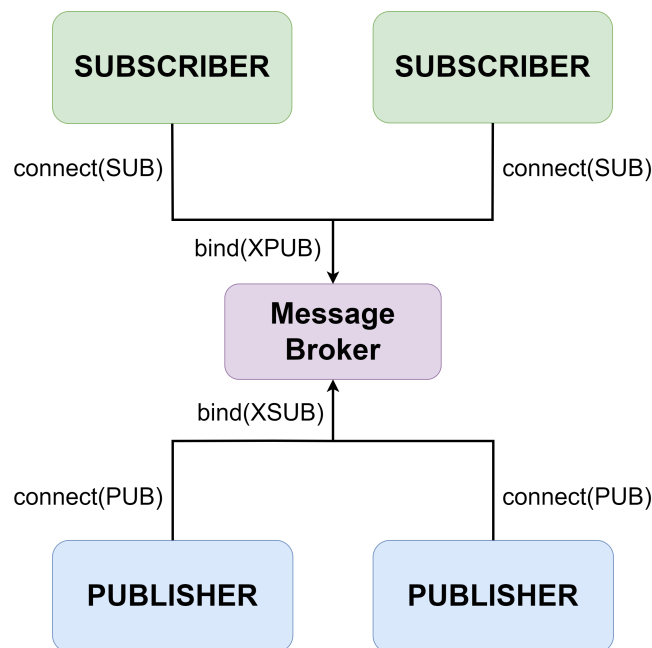


Figure 14: Implementation of an intermediary message broker in ZeroMQ

ZeroMQ is inherently brokerless, which provides some benefits in terms of performance, but there are use cases where an intermediary or proxy is helpful to reduce complexity. With the publish/subscribe pattern, we have publishers and subscriber clients. If we have a setup where we have one static publisher and several dynamic subscribers, we do not require an intermediary. The publisher can configure an endpoint that each subscriber can connect to, which is simple and easy to manage. The complexity comes when we need to have more than one publisher. Each subscriber and publisher would need to be coupled correctly, which introduces many challenges. This problem would be solved with an intermediary that works as a fixed point that the publishers and subscribers can connect to, which can be seen in figure 14. The responsibility of connecting clients will

be handled by the intermediary, reducing the responsibility of the other clients at the cost of some performance. This is similar to how the message broker works in MQTT. In figure 14 we see that the sockets types on the message broker are the XPUB and XSUB sockets, which are helpful when needing to filter messages. We can also see that the message broker uses the bind method while the other clients use the connect method, the recommended configuration for ZeroMQ.

When filtering messages in ZeroMQ, we can use what is called an ‘envelope’. Usually, all subscribers will receive the messages sent from the publisher they are subscribed to. However, if we only want to see specific messages, envelopes can be helpful. The envelope frame is a key located in front of the data frame which will be sent. It is similar to the topics found in MQTT and MQTT-SN and needs to be specified on the publisher when it publishes a message and the subscriber when it subscribes.

2.10.6 Previous Studies on ZeroMQ

A study from 2020 [11] has evaluated the performance of the Data Distribution Service (DDS) protocol [36], MQTT and ZeroMQ under use in different IoT traffic conditions. Both MQTT and DDS are considered industry standards regarding messaging protocols for use in IoT. The study has looked into how the protocols perform under different load conditions regarding latency and throughput. The tests were performed with simulations of realistic data flows found in IoT scenarios, including high-frequency, periodic, and sporadic data. The results show that DDS operated better with the highest throughput while testing with most of the high-frequency data-flow use cases. In one to many tests (one publisher, seven subscribers), ZeroMQ achieved higher throughput in all test cases. The results did show that for lower (64B) and medium (2KB) message sizes, ZeroMQ did perform slightly better than DDS in periodic data-flow tests when evaluating latency. With these message sizes, MQTT had much higher latency. At larger message sizes, MQTT performed better than DDS and ZeroMQ, with a flat latency while using increased publishing rates.

Another study from 2021 [37] has evaluated several group communication protocols and their scalability over synchronized cooperative broadcast. The protocols compared in the study include: NATS [38], DisService [39], GDEM and NORM with ZeroMQ [40]. The metrics used to evaluate the protocols are delivery ratio, latency, and bandwidth utilization. The scenario used for experimentation is the Anglova scenario [17], with four companies, each having 24 nodes available. The experiments involved four tests, where each new test increased the number of nodes with an additional company, meaning that the first test used 24 nodes while the last test used all 96 nodes. The results show that NORM with ZeroMQ performed best when comparing the delivery ratio. These tests were conducted with no cutoff time for message delivery. With a cutoff time of 5 or 10 seconds, ZeroMQ-NORM performed slightly below that of DisService and GDEM. Tests using latency as a metric shows that ZeroMQ-NORM is second best, behind DisService. Looking at bandwidth utilization, ZeroMQ performs second worst, behind NATS.

3 Development

In order to test the performance of the protocols in DIL networks, we have developed an analysis tool running in Python (version 3.8.10). The tool can run tests with a specific setup and produce logs stored for analysis. This allows us to analyze and evaluate the performance of each protocol on the application layer. The tool is also able to save network data with the use of the network analysis tool, tcpdump [41]. Tcpdump saves packet information in a PCAP file [42], which can later be analyzed in the network analysis tool, Wireshark [43]. The decision to make an analysis tool for this thesis was mainly to have a test setup that we could configure to measure what we needed from each protocol. This would also give us more control and allow us to gather data and make relevant statistics for our research. The work on this tool was started during the preparatory project [8], where we implemented support for the Tactical Broadband, SATCOM, NATO Narrowband, and CNR network models, as well as the MQTT protocol. We have expanded the tool to include 5G as a network model and the MQTT-SN and ZeroMQ protocol during this thesis.

We have looked at several other solutions that provided testing environments for the protocols we have investigated. However, the issues with these were mainly that they did not cover the range of protocols or provide statistics for the specific metrics we wanted to test. One example of a test tool that we looked at before deciding on developing our own was pymqttbench [44], but this tool only allows for testing of the MQTT protocol and does not emulate DIL networks.

We also looked at the Common Open Research Emulator (CORE), and Extendable Mobile Ad-hoc Network Emulator (EMANE), which is a combination of tools used for network simulation research in the defense sector [45]. These tools can simulate radio models and run scenarios similar to what we have used in our testing. The drawback of using CORE and EMANE is that the tool is large, complex, and requires a reasonable understanding of the hardware to configure the setup correctly. Instead, we have chosen to use the Netem tool as a basis for creating and emulating the network models tested in this thesis. This is because Netem is empirically and statistically correct [46]. The use of Netem helps support RQ1 in that we correctly evaluate the protocols with this as our basis of emulation.

During our research, we found that the MQTT-SN and ZeroMQ with PGM protocols had little adoption, meaning that we would need a custom solution for our experimentation. The solution we have developed has given us more control and allowed us to make sure that we can fulfill the test requirements for this thesis.

The analysis tool allows for the dynamic configuration of MQTT, MQTT-SN, and ZeroMQ clients. The tool initializes the emulation of the DIL networks (see section 4.1), which can be changed for each test. We have used the Pandas library in Python (version 1.4.0) [47] to store logs and create statistics during each test run. During our testing, the analysis tool has created large amounts of data, which is why we have used a database to store data from each of the tests. Lastly, we have also implemented a frontend application. Through the frontend, we can configure and view tests as they are running and compare results from previously run tests. Additional details about the analysis tool that has not been described in this chapter can be found in appendix A.

3.1 Technology

When building our tool, we had to decide on several different technologies. As we had started developing the tool in Python, we saw no reason to change it for this thesis. The Plotly Dash framework (version 2.1.0) [48] was used for developing the frontend and is purpose-made for creating applications used for data analysis. We have expanded our frontend with many new functionalities that were easy to implement using this frame-

work. To make the application look more professional and user-friendly, we have used Bootstrap with Plotly Dash (version 1.1.0), which introduces components that enhance these aspects. The incentive for creating a frontend for the analysis tool was mainly to show the statistics in a presentable and intuitive manner. This allows for easier comparisons between protocols and provides us with simple ways to export figures.

The database we have used to store data from each of the tests is MySQL (version 5.7.38) [49]. The database is hosted on an NTNU server running Ubuntu (version 0.16.04.1). This means that the database connection requires us to either be on a NTNU network or be connected to one through VPN. The database we have used is a relational database that suits the needs of our project, meaning that it supports the structure of the data we have generated during testing. This allows for more straightforward queries that can be handled on the database server, limiting the operations needing to be done on the client application. Some data storage also happens on the client, but these are primarily backups that mirror what is saved in the database. We have used the Pandas library, which is suitable for these types of tasks.

3.2 Supported Protocols

During this thesis, we have included two new protocols in our analysis system, namely MQTT-SN and ZeroMQ. We did have support for MQTT-SN in the preparatory project but have made several changes to allow for testing of the protocol.

With the inclusion of MQTT-SN, we have implemented a publisher and subscriber client that can send and receive messages over the network we have used to evaluate the protocols. We have two types of publisher and subscriber clients, as we wanted to test out two client implementations. The first one is a custom C to Python converted client that we have modified and used to fit our tool. The client has been modified so that it would be more similar to what we used when testing MQTT. Some limitations prevent us from having a more comparable client to MQTT, namely that it does not support QoS 2 and has fewer available statistics (number of disconnecting clients). The other MQTT-SN client type we have implemented in our tool is a Java client. This client is called using the subprocess module in Python. From initial testing, this client was found to be too slow, most likely due to subprocess calls, but we still have kept support for it.

The ZeroMQ clients we have implemented are also based on our MQTT clients as much as it was possible. We used the code recommended by the ZeroMQ developers to create the clients, with a few modifications to make it usable in our system. Both the TCP and PGM protocols use similar clients, where the only change is how they connect to the message broker.

3.3 Test Framework

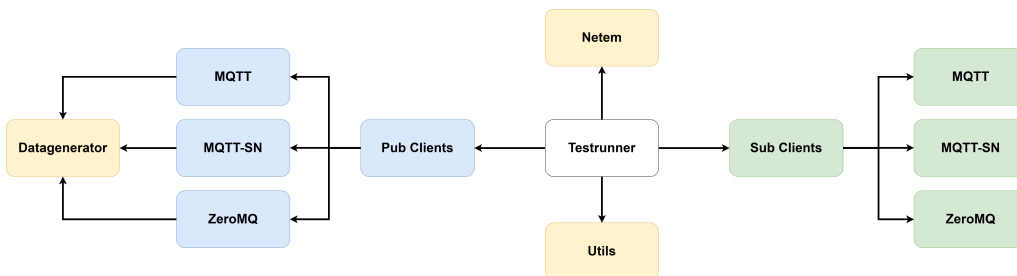


Figure 15: Architecture of the test framework

Figure 15 shows the components that compose the test framework. The testrunner component is responsible for configuring the clients and running the test based on the parameters that it is given. The testrunner uses the Netem networking tool [50] to limit the network based on the network model (section 4.1 describes Netem in detail). The given configuration parameters are stored in the database. Each client is separated into their own thread to run many clients in parallel. This allows us to simulate many devices sending and receiving messages simultaneously. The testrunner will initialize the correct number of publishers and subscribers based on the protocol type given as a parameter. The publishing clients will use the datagenerator component to create the data that will be exchanged. There is support for two types of messages, either GPS data with latitude, longitude, and altitude points or image data in the PNG format. In addition to this data, the datagenerator will add additional metadata that provides information about the message sent. This includes timestamps and message id, among other relevant metadata. Information related to the clients and the data sent is stored both locally and in the database. The testrunner will also initialize the subscriber clients, which will subscribe to the given topic. The utils components contain additional helpful methods for the testrunner component and include functions that store logs locally and create some statistics.

3.4 Frontend Application

Analysis tool
Test Results Page Test Configuration Page

Test Configuration

Protocol

MQTT x >

Network Model

TacticalBroadband x >

Data Type

GPS x >

Duration

150 x >

Publish Delay

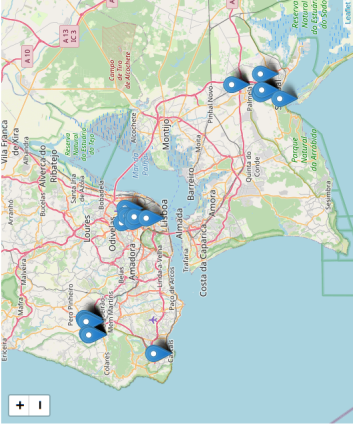
10 x >

Number of Nodes

27 x >

Start

GPS data



Statistics

QoS	Min Delay	Avg Delay	Max Delay
0	0.11	0.13	0.2
1	0.11	0.15	0.26
2	0.32	0.39	0.65

Data

Name	QoS	Message ID	Message Receive Time	Transmit Delay	Lat	Lon
SUB_USA_1	1	2	2022-06-09T08:45:47	0.12	38.52	-8.89
SUB_USA_1	1	1	2022-06-09T08:45:37	0.16	38.52	-8.89
SUB_USA_3	1	2	2022-06-09T08:45:47	0.13	38.52	-8.88
SUB_USA_3	1	1	2022-06-09T08:45:37	0.16	38.52	-8.88
SUB_USA_5	1	1	2022-06-09T08:45:37	0.14	38.52	-8.89
SUB_USA_5	1	2	2022-06-09T08:45:47	0.12	38.52	-8.89

Figure 16: Configuration page



Figure 17: Statistics page

During this thesis, the frontend application has been expanded to a multi-page application to minimize complexity and increase intuitiveness. There are now two pages, one containing the configuration as can be seen in figure 16, and one containing the results as can be seen in figure 17. We can select the parameters we want to use during a test on the configuration page, including the protocol, network model, data type, duration, publish delay (time between each sent message), and the number of nodes we want to use. Once the parameters have been set, we can run the test, sending the configuration parameters to the analysis framework which initializes the clients. As the test is running, we can view statistics from that test on the page. If the data type is GPS, the map will show the coordinates of each node. The map we have used is a Dash Leaflet component (version 0.1.23) [51]. The statistics table shows the available metrics we wish to look at including minimum, average, and max delay. The data table shows the data received from the subscriber nodes as they receive a packet. We can view and compare tests that we have previously run on the results page. Here we can load in tests and view them in various different charts. There is also a statistics table that shows the general statistics we want to look at on the application layer.

3.5 Data Storage

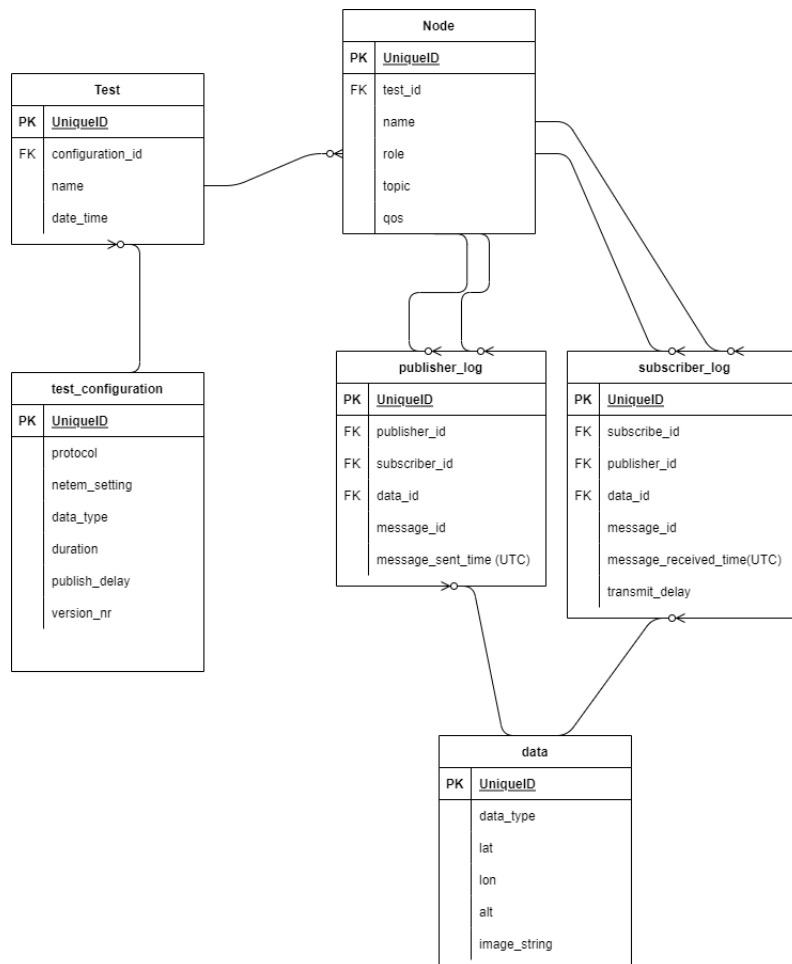


Figure 18: Structure of database in MySQL

Figure 18 shows the structure of the MySQL database, including the tables, their fields, and their relation. The database structure is identical to the one we have used in the preparatory project. To fetch data from the database, we can use join commands that give us relevant statistics that can be presented in the frontend application. Information

relevant to each test is stored in a table that has a relation to a separate table containing the test configuration. Every test will have nodes, meaning the clients used for the protocols. This node can either be of type publisher or subscriber and has relevant information about their name, topic, and QoS. Based on the node type, it will either relate to a publisher log or a subscriber log. The publisher log will contain information about who the receiving client is, the data's id, a message id, and the time when the message was sent. The data id references the data contained in the message, which is stored in a separate table (see section 4.5 for more on the data types). The subscriber log contains similar fields to that of the publisher log. The differences are that the subscriber log stores information about when the message was received and the calculated transmit time for the specific message.

3.6 Complete Architecture

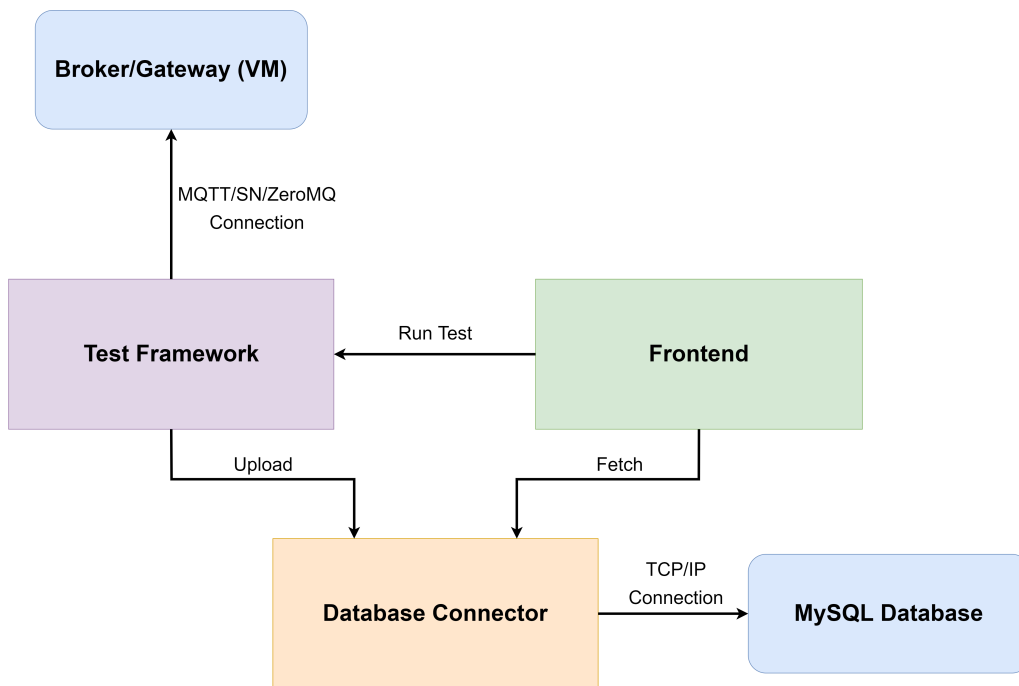


Figure 19: System overview

Figure 19 shows all the modules that make up the analysis tool and include: the test framework, frontend, and database connector. In addition, there is also the external MySQL database. From the figure, we can observe how the modules interact. The test framework contains a connection to the message broker (or, in the case of MQTT-SN, the gateway) that is located on the virtual machine (see section 4 for more on the virtual machine setup). To upload data to the database, the test framework uses several methods in the database connector. The configuration of each test is set in the frontend, which sends the configuration parameters to the test framework, which then runs the test. To get the queries used to fetch data from the database, the frontend also uses methods from the database connector. In addition to providing methods for uploading and getting queries, the database connector maintains the connection to the MySQL database.

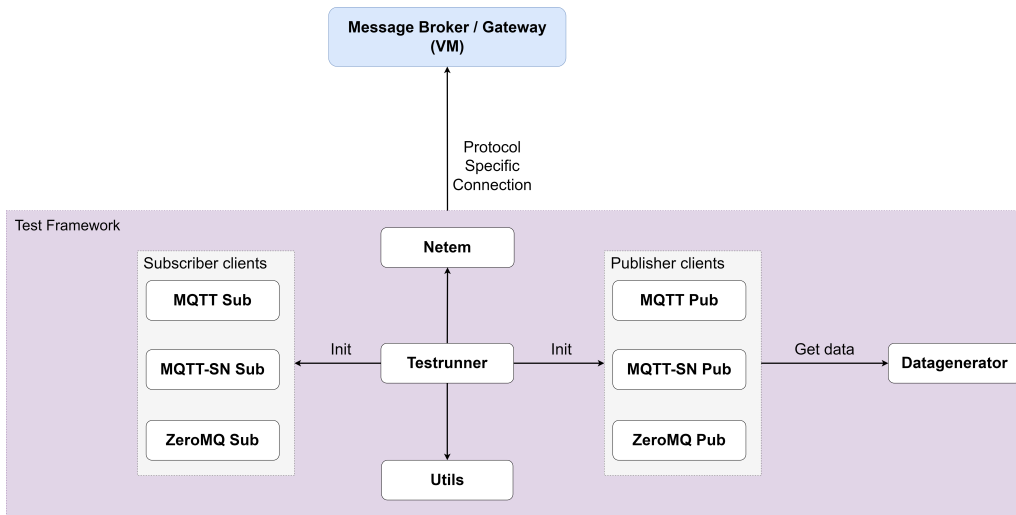


Figure 20: Overview of the test framework module

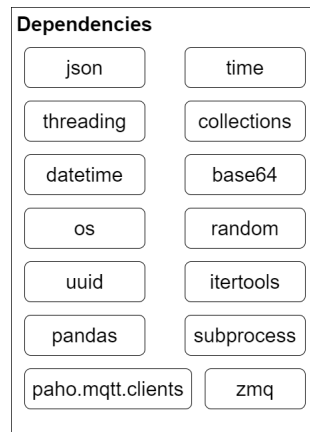


Figure 21: Dependencies of the test framework module

Figure 20 gives an overview of the test framework module and its components, while figure 21 shows its dependencies. Some dependencies are built-in in Python, while some are external (pandas, paho.mqtt.client, and zmq) and require installation via the Package Installer for Python (PIP). Note that there is no package for MQTT-SN as there does not exist one for Python.

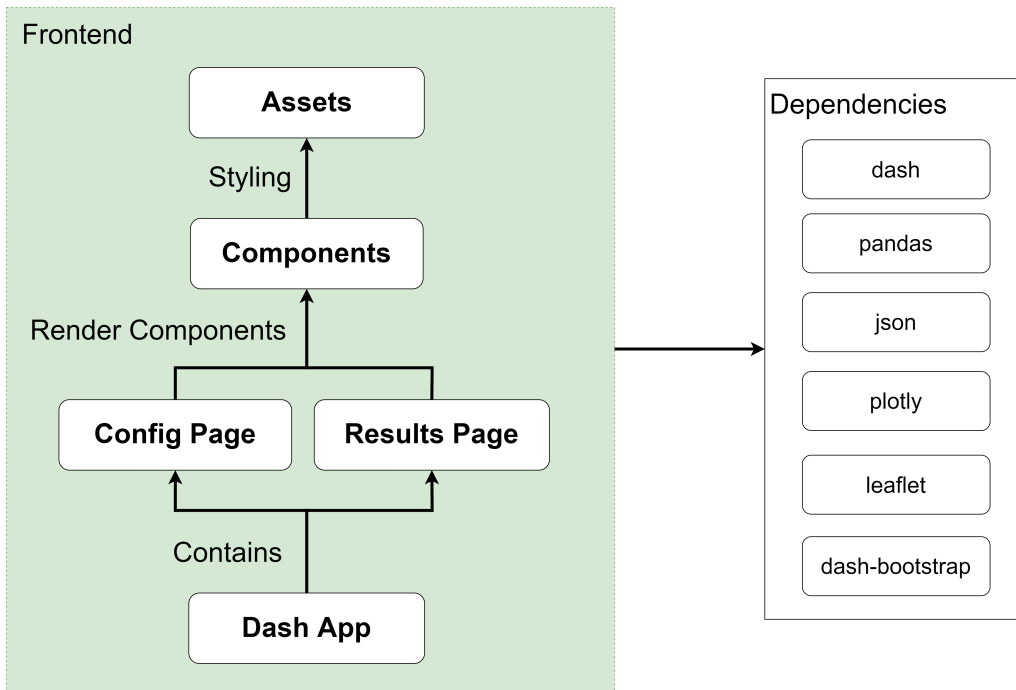


Figure 22: Overview of the frontend module

Figure 22 gives an overview of the frontend module, developed using the Plotly Dash framework. The application layout is rendered through the Dash App, which also instantiates the application. The Dash App contains the pages of the multi-page application. Each page is responsible for rendering the layout and have callback functions used to interact with the interface. The layout on each page imports different sections of the page from the Components and renders them. The Components are split up into different files, meaning that one file contains the graphs while another contains the statistics table. Some of the Components are styled (in addition to the styling provided by Bootstrap), and these styles are found in Assets. To visualize graphs, the frontend module uses Plotly which provides a range of different graph types that are simple to implement into the Dash application itself. Most of the interface and the tables use the Dash-Bootstrap library that provides pre-styled components. The map component uses Leaflet to display positional information in a map. The Pandas library is mainly used to manage the data fetched from the database.

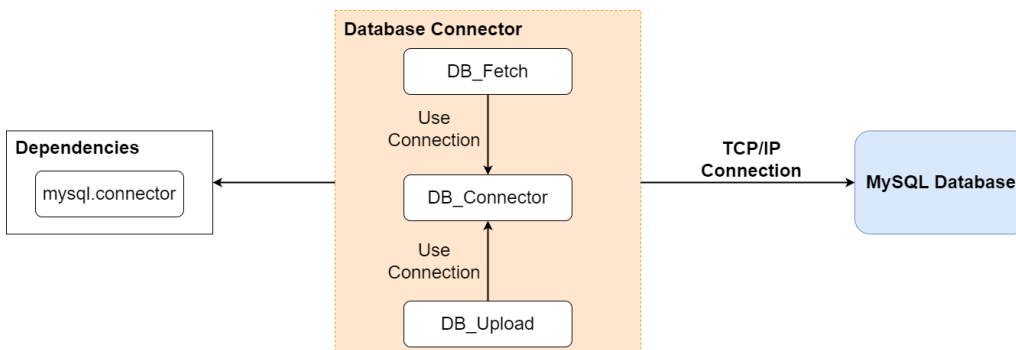


Figure 23: Overview of the database-connector module

Figure 23 shows the database connector module. We have used MySQL Connector (version 8.0.28) [52] to connect the frontend and the test framework to the database. DB_Fetch has methods for getting queries to be sent to the database, while DB_Upload has methods for posting data to the database. The DB_Fetch and the DB_Upload com-

ponents require the connection maintained in the DB_Connector module to query the database. The only dependency used in this module is mysql.connector.

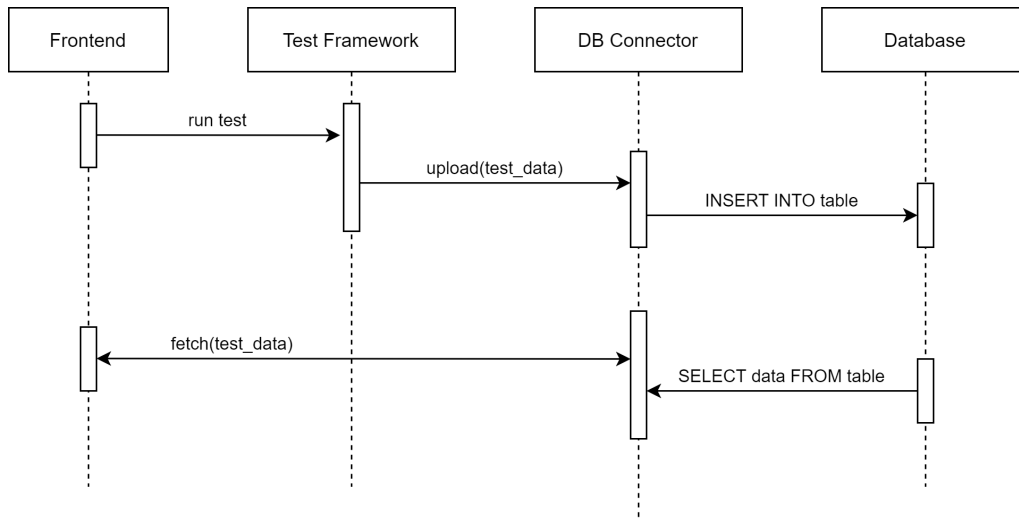


Figure 24: Sequence diagram showing the data flow through the modules

Figure 24 is a diagram that shows the frontend and test framework interacting with the database connector to upload and fetch data. The sequence begins with a command from the frontend to the test framework to start the test. The test framework will then configure and set up the clients, using the user's parameters in the frontend. Information about the test and the test configuration is then sent to the database through methods in the database connector. After a test has started, data will continuously be uploaded to the database from each client in the test framework. Finally, when data from a test populates the database, the frontend can fetch it through methods in the database connector.

4 Test Setup

In this study, we have tested the MQTT, MQTT-SN, and ZeroMQ protocols. The goal was to evaluate the performance of each of the protocols, with the main metrics being data rate, message size, packet loss, and transmit delay. We have used the analysis tool we have developed to run the tests, which allows for dynamic configurations for clients and network models. The setups for the clients are similar, but there are some changes due to differences in the protocols and certain limitations. To run the setup, we use a Linux machine running Ubuntu (version 20.04.3 LTS), where we can configure and run the tests. On the machine, we have set up a virtual machine (also running on Ubuntu), which works as a server that the clients can connect to when running a test. Having the virtual machine running on the same hardware, we can disregard time synchronization, which would be an issue if the clients needed to communicate across separate physical machines. The virtual machine has been set up using the Oracle VM Virtualbox (version 6.1) [53]. The specific settings used for the virtual machine can be found in appendix B.

We have used data that is similar to a previous study [12] in order to simulate a Blue Force Tracking (BFT) scenario. BFT is a system that provides situational awareness and gives an overview of the situation on the battlefield to commanders and troops [54]. In addition, we have also used image data which is useful in conveying information that words and numbers cannot, like objects or people. A more detailed description on data in section 4.5.

4.1 Netem

```
#!/bin/bash
echo "... to RESET do: sudo tc qdisc del dev vboxnet0 root"
sudo tc qdisc add dev vboxnet0 root Netem rate 9.6kbit loss 1% delay 100ms
```

Listing 1: Netem script for setting the CNR network model with 1% loss

In order to simulate the network models we wanted to test with, we have used the network emulation tool Netem. The tool has allowed us to configure the network connection between the host computer and the virtual machine, changing parameters like network delay, packet loss, and available bandwidth. This useful tool allows us to evaluate how the protocols perform in different environments. The Netem configurations for each test (see network details in table 1) are found in a shell script, which the analysis program will run at the start of each test. Once the test has concluded, the limitations are removed. Listing 1 shows the script that runs at the start of a test using the CNR network model with 1% loss. Here we see that `vboxnet0`, which is the interface to the virtual machine, is limited to a data rate of 9.6 kbit/s, with a 1% loss factor and a 100 millisecond delay. Similar settings are used to configure and simulate the remaining network models we wished to test. To validate that the Netem limitations are correctly set during a test, we have used `iPerf3` [55]. `iPerf3` is a network monitoring tool that, when used, showed that the limitations on the network were as they should be during testing. This tool has helped establish that the basis of our protocol evaluation is correct since it validates the use of Netem to emulate the network models, which is what we want to answer with RQ1. Appendix C has more detail on how we used `iPerf3`.

we also need clients that can send and receive MQTT-SN messages. For this, we have used a custom C to Python translated solution [59].

While testing the MQTT-SN protocol, we have used two test variations. The standard test variation is MQTT-SN to MQTT-SN, meaning that the publishers produce MQTT-SN messages, and the subscribers are configured to receive MQTT-SN messages. The other variation is a hybrid that is more common to see in other implementations. The hybrid variation has publishers sending MQTT-SN messages to the gateway, which then forwards them to the broker as MQTT messages. The subscribing clients are standard MQTT clients that will receive MQTT messages.

We encountered some issues trying to configure the clients and get them to work, which has resulted in some limitations when testing the protocol. The limitations include problems sending image data and using QoS level 2. Due to this, all tests are run using GPS data and QoS levels 0 and 1. In addition to this, we have only run the MQTT-SN tests using 18 clients instead of the 27 we have used for the other protocols. This was due to issues getting the clients to keep a stable connection with the gateway as they were being initialized. Otherwise, the configurations are the same, meaning that each test runs for 10 minutes with a publish delay of 10 seconds. Each network model has two tests, one for each variation.

4.4 ZeroMQ

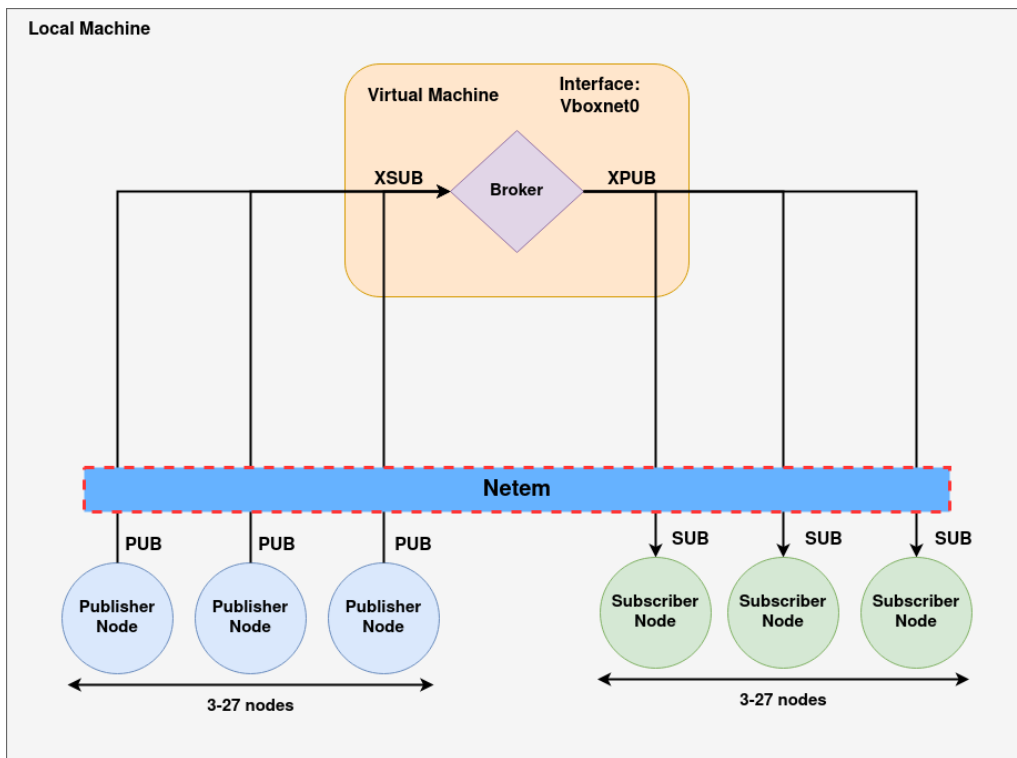


Figure 26: Test setup for ZeroMQ

Figure 26 show the setup we have used when testing with ZeroMQ. The setup was designed to replicate the tests we conducted using the MQTT and MQTT-SN protocols. The setup is configured to use 27 clients of both the subscriber- and publisher types, as we have used for MQTT. As previously mentioned, ZeroMQ does not provide out-of-the-box, ready-to-use clients. The client needs to be implemented by the developer and we have created our own clients that use the publish/subscribe pattern. The clients are inspired by patterns detailed in the ZeroMQ documentation [60]. Both clients are

lightweight, with minimal configuration changes aside from what is standard and what we require to gather statistics for testing.

```
import zmq

xpub_addr = 'tcp://*:5000'
xsub_addr = 'tcp://*:5001'
context = zmq.Context()

#create XPUB
xpub_socket = context.socket(zmq.XPUB)
xpub_socket.bind(xpub_addr)
#create XSUB
xsub_socket = context.socket(zmq.XSUB)
xsub_socket.bind(xsub_addr)

#create poller
poller = zmq.Poller()
poller.register(xpub_socket, zmq.POLLIN)
poller.register(xsub_socket, zmq.POLLIN)
print(zmq.zmq_version())

while True:
    # get event
    event = dict(poller.poll())
    print(event)
    if xpub_socket in event:
        message = xpub_socket.recv()
        print("[BROKER] xpub_socket recv message: %r" % message)
        xsub_socket.send(message)
    if xsub_socket in event:
        message = xsub_socket.recv()
        print("[BROKER] xsub_socket recv message: %r" % message)
        e = xpub_socket.send(message)
    print(e)
```

Listing 2: Code used for the message broker

We have developed our custom solution for the intermediary message broker that is as lightweight and simple as possible. This intermediary works by providing two endpoints using the special XSUB and XPUB sockets, which the clients can connect to. This intermediary is located on the virtual machine similar to the broker and gateway used by MQTT and MQTT-SN. Listing 2 shows the code to the message broker. The address for the special X.PUB and X.SUB sockets are configured with the TCP protocol, allowing connections from any address as long as it is on port 5000 (subscriber) or 5001 (publishers). The broker then polls for incoming messages, stored as events, and then sends the message as soon as possible to the corresponding receiver. The implementation we have used is based on one created by a developer on GitHub [61].

We have not only tested ZeroMQ using the TCP protocol but also PGM in order to test a protocol that supports reliable multicasting. The setup we have used is similar to the one we used for TCP, and we still utilize a message broker on the virtual machine where all messages are filtered. The only change needed is to configure ZeroMQ to use PGM. PGM requires the use of raw sockets, where raw sockets mean that one can determine all parts of a packet manually [62]. This is restricted in Linux, which can cause some issues with the implementation, and we have instead opted to use the encapsulated PGM protocol (EPGM). The only difference with EPGM is that the PGM packet is encapsulated into a UDP packet, meaning that we have circumvented the Linux restrictions. Details around how to configure ZeroMQ with PGM can be found in appendix E.

The settings used for testing are the same as with MQTT, meaning a test duration of 10 minutes and 10 seconds between each published message. ZeroMQ does not provide QoS levels like MQTT, which means that each network model has been tested one time using this setup.

4.5 Data

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": {
      "lat" : 10.704024011040783,
      "lon" : 59.9261147278993,
      "alt" : 0
    }
  },
  "properties": {
    "country": "NOR",
    "unit": "NOR-UNIT002",
    "node_name": "PUB_NOR_1",
    "node_id": 11,
    "msg_id": 12,
    "data_id" : 00c4901d-c6f2-4a69-8b76-58c58ce90211,
    "timestamp": 2021-11-30 09:32:36.435350
  }
}
```

Listing 3: Format of a message with GPS data that has been published

The messages that have been sent over the network contain data that simulates information that could be sent in real-life scenarios. We have in our testing used two data types, GPS and images. This was to compare the sending of smaller messages in the form of GPS data to larger ones in the form of images. Both civilian and military applications need to establish where people and objects are, making GPS data highly relevant for many scenarios. The use of image data is also relevant to provide additional information, either directly or through post-processing in the form of machine learning. The data was packaged in a JSON object [63], using the GeoJSON [64] format for transfer with the protocols. The image data we have used during testing comes in the PNG format with a resolution of 20 by 20 pixels and is encoded as a byte string and packaged inside the JSON message, alongside other relevant information. Listing 3 shows an example of how the data is structured. In this instance, the data is of the type GPS.

5 Results

This section describes our results while testing the MQTT, MQTT-SN, and ZeroMQ protocols. The IST-150 research group has previously conducted experiments with MQTT using the Tactical Broadband network model (both real and through emulation) [12]. Our results from the same tests support the findings from the research group, which solidifies the results from our testing and substantiates the answer to RQ1. This section presents the combined results from each protocol on the network and application layer, categorized by the data used during the test. For the complete test results, look at appendices F to I.

5.1 MQTT

Table 8 shows the gathered results from all tests with GPS data on the network layer. From the results we have some observations:

- The data rate decreases as the network models become more challenging. The CNR models have much lower data rates than the other tested models. Tactical Broadband has the highest data rates at 25 kbit/s, while both CNR models have the lowest at 10 kbit/s.
- The message sizes remain the same size for Tactical Broadband, SATCOM, and NATO Narrowband tests. The sizes increase drastically when using CNR with 1% and 10% loss.
- Retransmissions and duplicate acknowledgments happen rarely compared to the number of packets sent using the Tactical Broadband and SATCOM network models. There is a slight increase using NATO Narrowband, but the most significant increase happens with the CNR models, which both have a high amount of retransmissions and duplicate acknowledgments.

Network model	Data rate	Avg msg size	TCP retransmissions	Dup ACK
5G	24 kbit/s	482 bytes	0	1
Tactical broadband	25 kbit/s	482 bytes	82	27
SATCOM	23 kbit/s	482 bytes	28	27
NATO narrowband	22 kbit/s	482 bytes	313	313
CNR 1% loss	10 kbit/s	718 bytes	1957	713
CNR 10% loss	10 kbit/s	669 bytes	1729	560

Table 8: Results from all standard tests using GPS data on the network layer

Table 9 shows the gathered results from all tests with image data on the network layer. From the results we have some observations:

- The 5G network model test had the highest data rate at 69 kbit/s. The following tests showed a decline in data rates, with NATO Narrowband and CNR having the lowest.
- The message sizes vary significantly in the different network model tests. The 5G network model test had the highest at 1553 bytes, while SATCOM had the lowest at 467 bytes.
- The 5G, Tactical Broadband, and SATCOM tests caused very few retransmissions and duplicate acknowledgments, while NATO Narrowband and CNR tests caused many retransmissions and duplicate acknowledgments. During the test with 5G, there were zero retransmissions and duplicate acknowledgments.

Network model	Data rate	Avg msg size	TCP retransmissions	Dup ACK
5G	69 kbit/s	1553 bytes	0	0
Tactical broadband	40 kbit/s	521 bytes	121	61
SATCOM	22 kbit/s	467 bytes	162	167
NATO narrowband	4.28 kbit/s	1078 bytes	1424	341
CNR 1% loss	2.64 kbit/s	1041 bytes	1406	347
CNR 10% loss	2.84 kbit/s	920 bytes	1596	330

Table 9: Results from all standard tests using image data on the network layer

Table 10 shows the gathered results from all tests with GPS data on the application layer. From the results we have some observations:

- The 5G, Tactical Broadband, SATCOM, and NATO Narrowband network models had no issue transmitting messages and recorded zero packet loss.
- QoS 0 and 1 had the same or similar transmit delay using the 5G, Tactical Broadband, and SATCOM network models. QoS 2 had a much larger transmit delay with these models compared to QoS 0 and QoS 1.
- CNR with 1% and 10% loss caused issues with message transmissions for all QoS levels.
- QoS 1 saw a large increase in packet loss from the CNR test with 1% loss, where it was zero, to the test with 10% loss. QoS 0 had a reduction in the amount of packet loss.
- QoS 0 had much lower transmit delays than QoS 1 and QoS 2 when using the CNR network models. QoS 2 had the highest transmit delays, while QoS 1 was slightly lower but still high.
- The number of disconnected clients increased with the QoS level on the CNR test with a 10% loss.

Network model	QoS	Messages Sent	Packet loss	Min delay	Avg delay	Max delay	Disconnections
5G	0	536	0%	0.04 s	0.07 s	0.99 s	0
	1	532	0%	0.04 s	0.07 s	1.23 s	0
	2	531	0%	0.04 s	0.09 s	0.63 s	0
Tactical broadband	0	540	0%	0.11 s	0.12 s	0.45 s	0
	1	540	0%	0.11 s	0.12 s	0.44 s	0
	2	540	0%	0.31 s	0.33 s	0.74 s	0
SATCOM	0	540	0%	0.57 s	0.60 s	1.16 s	0
	1	540	0%	0.57 s	0.62 s	0.74 s	0
	2	539	0%	1.69 s	1.78 s	2.41 s	0
NATO narrowband	0	540	0%	1.24 s	3.92 s	10.02 s	0
	1	533	0%	2.62 s	5.31 s	12.45 s	0
	2	532	0%	9.55 s	19.00 s	40.25 s	0
CNR 1% loss	0	357	75 (21.01%)	0.62 s	28.58 s	94.43 s	17
	1	214	0%	3.37 s	146.39 s	512.25 s	18
	2	258	60 (23.26%)	13.67 s	226.88 s	610.17 s	18
CNR 10% loss	0	421	36 (8.55%)	0.55 s	33.80 s	120.18 s	9
	1	315	61 (19.37%)	1.19 s	82.14 s	500.62 s	15
	2	259	160 (61.78%)	4.10 s	85.05 s	185.67 s	18

Table 10: Results from all tests using GPS data on the application layer

Table 11 shows the gathered results from all tests with image data on the application layer. From the results, we have some observations:

- Message transmission went without any issues, and there was no recorded packet loss when testing with 5G, Tactical Broadband, and SATCOM network models.

- The transmit delay was similar to when testing with GPS data for the 5G and Tactical Broadband network models.
- The packet loss and transmit delays were significant for tests with NATO Narrowband and CNR with both 1% and 10% loss. Very few messages were sent, and most were lost during these tests, where the lowest packet loss was 85.86% with QoS 0 using the NATO Narrowband network model.
- Tests using NATO Narrowband and CNR had many clients disconnect.

Network model	QoS	Messages Sent	Packet loss	Min delay	Avg delay	Max delay	Disconnections
5G	0	532	0%	0.04 s	0.09 s	1.32 s	0
	1	533	0%	0.04 s	0.08 s	1.18 s	0
	2	533	0%	0.04 s	0.11 s	2.21 s	0
Tactical broadband	0	540	0%	0.11 s	0.15 s	0.74 s	0
	1	540	0%	0.11 s	0.16 s	0.58 s	0
	2	539	0%	0.32 s	0.41 s	1.44 s	0
SATCOM	0	540	0%	0.63 s	2.20 s	4.35 s	0
	1	540	0%	0.61 s	1.86 s	4.38 s	0
	2	539	0%	3.16 s	4.34 s	5.54 s	0
NATO narrowband	0	99	85 (85.86%)	1.51 s	23.05 s	50.16 s	18
	1	101	92 (91.09%)	17.87 s	24.79 s	33.59 s	18
	2	108	108 (100%)	NA	NA	NA	18
CNR 1% loss	0	104	95 (91.35%)	6.97 s	35.22 s	59.02 s	18
	1	106	97 (91.51%)	5.58 s	34.57 s	57.64 s	18
	2	108	108 (100%)	NA	NA	NA	18
CNR 10% loss	0	105	99 (94.29%)	5.30 s	30.67 s	49.63 s	18
	1	106	97 (91.51%)	6.66 s	40.10 s	58.87 s	18
	2	105	105 (100%)	NA	NA	NA	18

Table 11: Results from all tests using image data on the application layer

5.2 MQTT-SN

Table 12 shows the gathered results from all standard tests on the network layer. From these results, we can observe that:

- The tests from Tactical Broadband to CNR with 1% loss had identical data rates at 13 kbit/s. 5G had a data rate that was slightly lower at 12 kbit/s, while CNR with 10% loss had a much lower rate at 4.09 kbit/s.
- Average packet size remained the same across all tests at 432 bytes.

Network model	Data rate	Avg message size
5G	12 kbit/s	432 bytes
Tactical Broadband	13 kbit/s	432 bytes
SATCOM	13 kbit/s	432 bytes
NATO Narrowband	13 kbit/s	432 bytes
CNR 1% loss	13 kbit/s	432 bytes
CNR 10% loss	4.09 kbit/s	432 bytes

Table 12: Results from all tests using the standard implementation on the network layer

Table 13 shows the combined results for all hybrid tests (the hybrid concept was discussed in section 4.3) on the network layer. From these results we can observe that:

- The data rate was consistently higher for MQTT than MQTT-SN. The highest data rate from MQTT was 7.56 kbit/s, while the highest from MQTT-SN was 6.69 kbit/s.
- MQTT had consistently larger average message sizes than MQTT-SN, which is to be expected as the purpose of MQTT-SN is to reduce the overall message size.

Network model	Data rate		Avg message size	
	MQTT	MQTT-SN	MQTT	MQTT-SN
5G	7.35 kbit/s	6.69 kbit/s	484 bytes	432 bytes
Tactical Broadband	7.55 kbit/s	6.48 kbit/s	484 bytes	432 bytes
SATCOM	7.05 kbit/s	6.49 kbit/s	484 bytes	432 bytes
NATO Narrowband	7.20 kbit/s	6.47 kbit/s	484 bytes	432 bytes
CNR 1% loss	7.56 kbit/s	6.54 kbit/s	484 bytes	432 bytes
CNR 10% loss	4.23 kbit/s	4.42 kbit/s	483 bytes	432 bytes

Table 13: Results from all tests using the hybrid implementation on the network layer

Table 14 shows the combined results for the standard tests on the application layer. From these results, we can observe that:

- There is some variation in the number of messages sent with each protocol. Most tests resulted in around 540 messages being sent. The tests with CNR had the lowest amount of packets sent at 420.
- The tests with the 5G, Tactical Broadband, SATCOM, and NATO Narrowband network models had no issue transmitting messages in regards to packet loss. The packet loss was zero across all these tests.
- The tests using the CNR network model with 1% and 10% were the only tests that had packet loss. The packet loss was not high with 1% loss, where only six packets were lost with QoS 0 and 4 packets lost with QoS 1. The test using 10% loss had a packet loss of 100%, which is significantly more than any other test.
- The transmit delay was low across all tests, excluding CNR with 10% loss where no messages were received. The lowest delay was found using the 5G network model, with an average delay of 0.09 seconds. The highest delay was found using CNR with 1% loss, at 4.39 seconds using QoS 1.
- QoS 0 had lower average delays than QoS 1 during all tests.

Network model	QoS	Messages sent	Packet loss	Min delay	Avg delay	Max delay
5G	0	540	0%	0.07 s	0.09 s	1.27 s
	1	540	0%	0.07 s	0.10 s	0.82 s
Tactical Broadband	0	540	0%	0.15 s	0.17 s	1.31 s
	1	538	0%	0.15 s	0.19 s	0.91 s
SATCOM	0	540	0%	0.57 s	0.59 s	0.92 s
	1	540	0%	0.57 s	0.60 s	0.70 s
NATO Narrowband	0	540	0%	0.72 s	1.68 s	3.64 s
	1	540	0%	1.35 s	2.95 s	4.20 s
CNR 1% loss	0	540	6 (1.11%)	0.47 s	2.09 s	4.38 s
	1	540	4 (0.74%)	1.24 s	4.39 s	6.40 s
CNR 10% loss	0	420	420 (100%)	NA	NA	NA
	1	420	420 (100%)	NA	NA	NA

Table 14: Results from all tests using the standard implementation on the application layer

Table 15 shows the combined results for the hybrid tests on the application layer. From these results, we can observe that:

- Most tests had 540 messages sent, except Tactical Broadband which sent 538 messages and CNR with 10% loss, which only sent 480 messages using QoS 0 and 360 messages using QoS 1.
- As with the standard tests, only the CNR network model tests had packet loss. During the test using CNR with 1% loss, QoS 0 lost only 5 messages, and QoS 1 lost 6. The packet loss increased while testing using CNR with 10% loss, where QoS 0 had a packet loss of 20.21%, and QoS 1 had a packet loss of 25%.
- The transmit delays were low across all tests. The 5G test had the lowest average delay with 0.03 seconds, while CNR with 1% loss had the highest with 4.77 seconds.
- During all tests except for 5G, QoS 0 had lower average delays than QoS 1.

Network model	QoS	Messages sent	Packet loss	Min delay	Avg delay	Max delay
5G	0	540	0%	0.02 s	0.03 s	0.23 s
	1	540	0%	0.02 s	0.03 s	0.12 s
Tactical Broadband	0	540	0%	0.15 s	0.16 s	0.51 s
	1	538	0%	0.15 s	0.17 s	0.44 s
SATCOM	0	540	0%	0.57 s	0.61 s	1.02 s
	1	540	0%	0.57 s	0.62 s	1.02 s
NATO Narrowband	0	540	0%	0.72 s	2.00 s	4.13 s
	1	540	0%	1.04 s	2.79 s	4.92 s
CNR 1% loss	0	540	5 (0.93%)	0.47 s	2.17 s	7.12 s
	1	540	6 (1.11%)	2.53 s	4.77 s	8.12 s
CNR 10% loss	0	480	97 (20.21%)	0.46 s	1.67 s	5.78 s
	1	360	90 (25%)	1.10 s	3.13 s	5.00 s

Table 15: Results from all tests using the hybrid implementation on the application layer

5.3 ZeroMQ with TCP

Table 16 shows the combined results from all tests using GPS data on the network layer. From these results we can see that:

- The highest data rate was achieved during the tests with the 5G and Tactical Broadband network models, at 20 kbit/s. SATCOM had a data rate that was slightly lower at 19 kbit/s, while the tests with NATO Narrowband and CNR network models had significantly lower at 9.80 kbit/s, 8.51 kbit/s, and 9.58 kbit/s.
- The 5G, Tactical Broadband, SATCOM, and NATO Narrowband tests had the same message size of 488 bytes. CNR with 1% loss had a size that was 5 bytes bigger at 493 bytes, and CNR with 10% loss had a size that was 14 bytes higher at 532 bytes.
- 5G, Tactical Broadband and SATCOM had no significant amount of retransmissions or duplicate acknowledgments. NATO Narrowband and CNR with 1% loss had higher amounts, while CNR with 10% loss had the highest.

Network model	Data rate	Avg msg size	TCP retransmissions	Dup ACK
5G	20 kbit/s	488 bytes	0	0
Tactical broadband	20 kbit/s	488 bytes	24	0
SATCOM	19 kbit/s	488 bytes	0	0
NATO narrowband	9.80 kbit/s	488 bytes	95	137
CNR 1% loss	8.51 kbit/s	493 bytes	154	256
CNR 10% loss	9.58 kbit/s	532 bytes	232	471

Table 16: Results from tests using GPS data on the network layer

Table 17 shows the combined results from all tests using image data on the network layer. From these results we see that:

- The SATCOM network model had the highest data rate at 20 kbit/s. 5G and Tactical Broadband had slightly lower data rates at 19 kbit/s and 18 kbit/s. These results were significantly higher than that of other models. CNR with 1% had a much higher data rate than NATO Narrowband and CNR with 10% loss, at 12 kbit/s. The NATO Narrowband network model had the lowest data rate at 3.69 kbit/s.
- There is some variation in message sizes from all of the tests. The CNR model with 1% loss had the largest average message size at 1359 bytes, while 5G had the lowest at 497 bytes. The Tactical Broadband and SATCOM network models had slightly larger average message sizes at 500 and 506 bytes.
- Retransmissions and duplicate acknowledgments were low using the 5G, Tactical Broadband, and SATCOM network models. The numbers increased using the NATO Narrowband and CNR with 1% loss network models, while the CNR model with 10% loss had the highest overall.

Network model	Data rate	Avg msg size	TCP retransmissions	Dup ACK
5G	19 kbit/s	497 bytes	0	0
Tactical broadband	18 kbit/s	500 bytes	73	38
SATCOM	20 kbit/s	506 bytes	71	71
NATO narrowband	3.69 kbit/s	1044 bytes	242	248
CNR 1% loss	12 kbit/s	1359 bytes	272	340
CNR 10% loss	3.94 kbit/s	1166 bytes	335	286

Table 17: Results from tests using image data on the network layer

Table 18 shows the combined results from all tests using GPS data on the application layer. From these results, we see that:

- All tests sent the same number of messages at 1593.
- The 5G, Tactical Broadband, SATCOM, and NATO Narrowband network models had zero packet loss during their tests. Both CNR tests had the same packet loss, where 133 messages were lost.
- The transmit delay was relatively low using the 5G, Tactical Broadband, and SATCOM network models. The test using 5G had the lowest delay at 0.12 seconds. The delay increased in the CNR tests. The highest delay was 54.34 seconds using CNR with 10% loss, which is slightly higher than the test using CNR with 1% loss, which was 45.72 seconds.

Network model	Messages sent	Packet loss	Min delay	Avg delay	Max delay
5G	1593	0%	0.03 s	0.12 s	0.39 s
Tactical broadband	1593	0%	0.11 s	0.21 s	0.73 s
SATCOM	1593	0%	0.58 s	0.66 s	1.06 s
NATO narrowband	1593	0%	0.76 s	2.83 s	5.81 s
CNR 1% loss	1593	133 (8.35%)	0.91 s	45.72 s	162.75 s
CNR 10% loss	1593	133 (8.35%)	5.73 s	54.34 s	185.58 s

Table 18: Results from all tests using GPS data on the application layer

Table 19 shows the combined results from all tests using image data on the application layer. From these results, we see that:

- All tests were able to send 1566 messages each.
- The 5G, Tactical Broadband, and SATCOM models had zero packet loss, and the packet loss was much higher using the NATO Narrowband and CNR network models. Of these, NATO Narrowband had the lowest packet loss where 76.56% of the messages were lost, and CNR with 10% loss had the highest at 97.30%.
- The transmit delay for the 5G, Tactical Broadband, and SATCOM network models were relatively low, and 5G was the quickest at 0.36 seconds. The delay was much higher with the other network models, with the highest being NATO Narrowband, with an average delay of 269.58 seconds.

Network model	Messages sent	Packet loss	Min delay	Avg delay	Max delay
5G	1566	0%	0.05 s	0.36 s	0.75 s
Tactical broadband	1566	0%	0.14 s	0.47 s	1.41 s
SATCOM	1566	0%	0.63 s	1.82 s	3.59 s
NATO narrowband	1566	1199 (76.56%)	1.54 s	269.58 s	515.58 s
CNR 1% loss	1566	1488 (95.02%)	6.88 s	56.94 s	139.07 s
CNR 10% loss	1566	1514 (97.30%)	7.67 s	89.60 s	325.54 s

Table 19: Results from all tests using image data on the application layer

5.4 ZeroMQ with PGM

Table 20 shows the combined results from all tests using GPS data on the network layer. From these results we can see that:

- The highest data rate was found using CNR with 10% loss, at 52 kbit/s. 5G had a data rate of 28 kbit/s, significantly lower than the other models, where all had data rates above 40 kbit/s.
- The 5G, Tactical Broadband, and SATCOM network models had the same average message size of 491 bytes. NATO Narrowband had the largest average message size at 523 bytes. CNR with 1% had a size of 492 bytes, while CNR with 10% loss had a size of 493 bytes.

Network model	Data rate	Avg message size
5G	28 kbit/s	491 bytes
Tactical Broadband	42 kbit/s	491 bytes
SATCOM	40 kbit/s	491 bytes
NATO Narrowband	45 kbit/s	523 bytes
CNR 1% loss	49 kbit/s	492 bytes
CNR 10% loss	52 kbit/s	493 bytes

Table 20: Results from all tests using GPS data on the network layer

Table 21 shows the combined results from all tests using image data on the network layer. From these results we see that:

- 5G had the highest data rate at 637 kbit/s. As the networks became more challenging, the data rates decreased significantly. CNR with 1% loss had the lowest data rate at 27 kbit/s.
- As with the data rates, the average message size decreased when the networks became more challenging. 5G had the largest average size at 1310 bytes, while CNR with 10% loss had the lowest at 183 bytes.

Network model	Data rate	Avg message size
5G	637 kbit/s	1310 bytes
Tactical Broadband	502 kbit/s	1184 bytes
SATCOM	263 kbit/s	1136 bytes
NATO Narrowband	72 kbit/s	186 bytes
CNR 1% loss	27 kbit/s	208 bytes
CNR 10% loss	31 kbit/s	183 bytes

Table 21: Results from all tests using image data on the network layer

Table 22 shows the combined results from all tests using GPS data on the application layer. From these results we see that:

- The number of messages sent varies greatly from test to test. 5G had the highest number of messages sent at 1619, while CNR with 10% loss had the lowest with 1565 messages sent.
- The 5G, Tactical Broadband, and SATCOM network models lost no messages during their tests. NATO Narrowband had a packet loss just below 50%. CNR with 1% loss had the highest packet loss at 73.88%.
- The average delay was low while using the 5G, Tactical Broadband, and SACOM network models. 5G had the lowest delay at 0.10 seconds. NATO Narrowband and the CNR models had a significantly higher delay. The CNR models transmitted at just below 2 minutes, where CNR with 1% loss had the highest delay at 115.23 seconds.

Network model	Messages sent	Packet loss	Min delay	Avg delay	Max delay
5G	1619	0%	0.03 s	0.10 s	1.03 s
Tactical Broadband	1568	0%	0.15 s	0.31 s	1.94 s
SATCOM	1584	0%	0.58 s	0.72 s	1.10 s
NATO Narrowband	1592	770 (48.37%)	0.89 s	66.17 s	98.18 s
CNR 1% loss	1566	1157 (73.88%)	1.75 s	115.23 s	224.01 s
CNR 10% loss	1565	954 (60.96%)	1.30 s	102.75 s	140.94 s

Table 22: Results from all tests using GPS on the application layer

Table 23 shows the combined results from all tests using image data on the application layer. From these results we see that:

- As with GPS data, the number of messages sent from each test varies greatly. 5G sent the most messages at 1593, while CNR, with a 1% loss, sent the least by only 1522.
- We can see that only the 5G network model test had no packet loss. Tactical Broadband did not have a significant loss since only two messages went missing. SATCOM had some loss at 12.77%, while the NATO Narrowband and CNR tests had most of the messages missing.
- We can see that the delay was considerable regardless of the network model. Tactical Broadband had the lowest delay at 1.67 seconds, while CNR with 1% loss had the highest delay at 201.47 seconds.

Network model	Messages sent	Packet loss	Min delay	Avg delay	Max delay
5G	1593	0%	0.04 s	3.07 s	8.51 s
Tactical Broadband	1543	2 (0.13%)	0.18 s	1.67 s	6.56 s
SATCOM	1566	200 (12.77%)	0.84 s	14.76 s	56.52 s
NATO Narrowband	1540	1378 (89.48%)	1.70 s	116.62 s	222.08 s
CNR 1% loss	1522	1397 (91.79%)	14.93 s	201.47 s	353.92 s
CNR 10% loss	1532	1429 (93.28%)	9.98 s	196.52 s	348.02 s

Table 23: Results from all tests using image on the application layer

6 Analysis

Using the analysis tool we developed, we tested the MQTT, MQTT-SN, and ZeroMQ protocols using different network models. The main objective was to test these protocols in how they performed in networks with varying degrees of limitations. This section will analyze all protocols categorized by the network model. The results will focus on data gathered from the application layer; namely, average transmit delay and packet loss.

6.1 5G

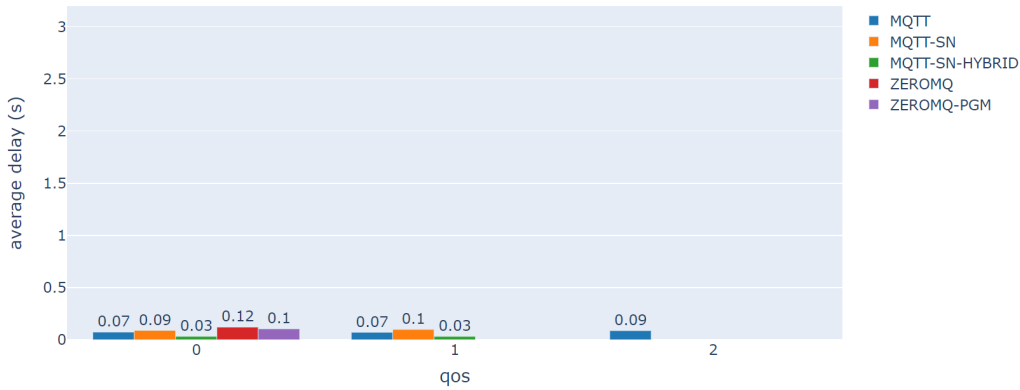


Figure 27: Average transmit delay times using the 5G network model with GPS data

Figure 27 shows how the protocols performed regarding average transmit delay during the tests with the 5G network model using GPS data. Note that both tests with ZeroMQ are positioned in the QoS 0 category. This will be the same for all the coming figures. Table 24 shows the packet loss for each protocol. Most of the protocols had low delays, at around 100 milliseconds per message. The protocol that stands out most is MQTT-SN with the hybrid variation, which is significantly quicker than the others. This was likely due to UDP, which has much less overhead than TCP. The standard variation is in line with the other protocol, so there might be some delay introduced by the clients used since this variation uses both MQTT-SN clients for publishing and subscribing. Looking at the QoS levels for MQTT and MQTT-SN, we see that the higher QoS levels had no significant impact on the average delay of the protocols, which is to be expected in this type of network.

Protocol	QoS	Packet Loss
MQTT	0	0%
	1	0%
	2	0%
MQTT-SN Standard	0	0%
	1	0%
MQTT-SN Hybrid	0	0%
	1	0%
ZeroMQ TCP	None	0%
ZeroMQ PGM	None	0%

Table 24: Packet loss from tests with the 5G network model using GPS data

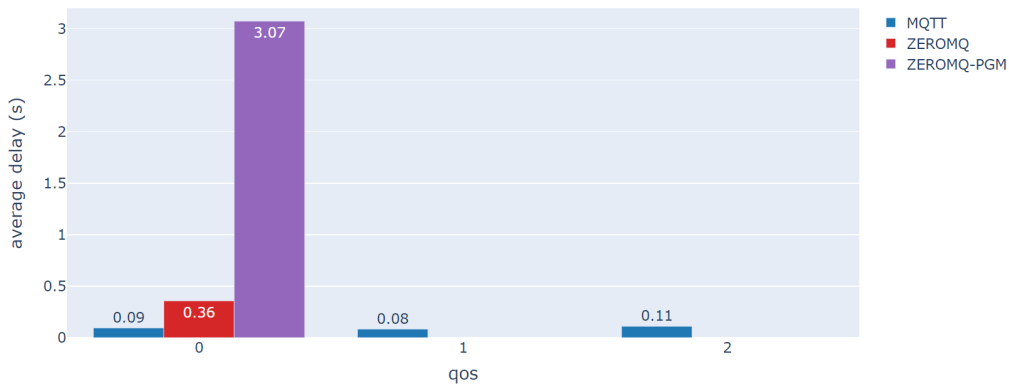


Figure 28: Average transmit delay times using the 5G network model with image data

Figure 28 shows how the protocols performed regarding average transmit delay during the tests with the 5G network model using image data. Note that since we did not perform tests with MQTT-SN using image data, the figure shows no results for this protocol. Table 25 shows the packet loss for each protocol. Here we see that the MQTT protocol had marginally higher delays than the tests with GPS data, which is expected due to the increase in message size. The ZeroMQ protocol showed a large increase in delay times for the TCP and PGM tests. The increased message size seems to have a greater effect while sending with these protocols than with MQTT, especially for PGM. The network statistics show that the number of packets created by multicasting is high, which may have affected performance somewhat.

Protocol	QoS	Packet Loss
MQTT	0	0%
	1	0%
	2	0%
MQTT-SN Standard	0	NA
	1	NA
MQTT-SN Hybrid	0	NA
	1	NA
ZeroMQ TCP	None	0%
ZeroMQ PGM	None	0%

Table 25: Packet loss from tests with the 5G network model using image data

6.2 Tactical Broadband

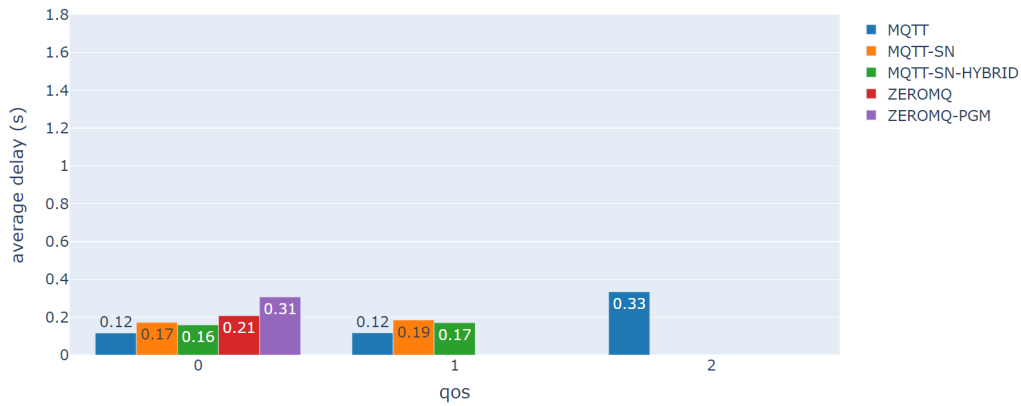


Figure 29: Average transmit delay times using the Tactical Broadband network model with GPS data

Figure 29 shows how the protocols performed in regards to average transmit delay during the tests with the Tactical Broadband network model using GPS data. Table 26 shows the packet loss for each protocol. Here we see similar results to that of the 5G tests, where the protocols had mostly similar delay times, with MQTT-SN using the hybrid variation being the quickest. We can see that using MQTT with QoS 2 had the highest delay, which had a bigger effect using this network model than with the 5G model. We saw an increase in the number of retransmissions and duplicate acknowledgements during this test which might indicate that some of the messages had issues during transmission.

Protocol	QoS	Packet Loss
MQTT	0	0%
	1	0%
	2	0%
MQTT-SN Standard	0	0%
	1	0%
MQTT-SN Hybrid	0	0%
	1	0%
ZeroMQ TCP	None	0%
ZeroMQ PGM	None	0%

Table 26: Packet loss from tests with the Tactical Broadband network model using GPS data

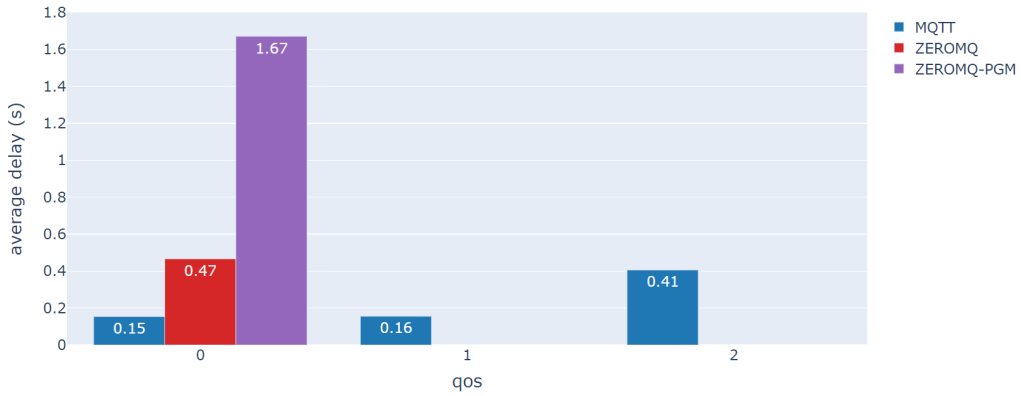


Figure 30: Average transmit delay times using the Tactical Broadband network model with image data

Figure 30 shows how the protocols performed in regards to average transmit delay during the tests with the Tactical Broadband network model using image data. Table 27 shows the packet loss for each protocol. We again see that ZeroMQ performed worse than MQTT, at least with QoS 0 and 1. QoS 2 had a slightly higher average transmit time, indicating that the reassurance introduced by this QoS level has a significant impact on performance. The test using ZeroMQ with PGM had a significantly higher average delay than that of MQTT and ZeroMQ with TCP. Comparing this test using PGM to the one done with the 5G network model, we can see that the delay is significantly lower. This was unusual since the 5G model is not as restricted as the Tactical Broadband model. Here we can see that ZeroMQ with PGM has some packet loss. Only 0.13% of the messages were lost, but this was unusual to see in a network of this type from our experience.

Protocol	QoS	Packet Loss
MQTT	0	0%
	1	0%
	2	0%
MQTT-SN Standard	0	NA
	1	NA
MQTT-SN Hybrid	0	NA
	1	NA
ZeroMQ TCP	None	0%
ZeroMQ PGM	None	0.13%

Table 27: Packet loss from tests with the Tactical Broadband network model using image data

6.3 SATCOM

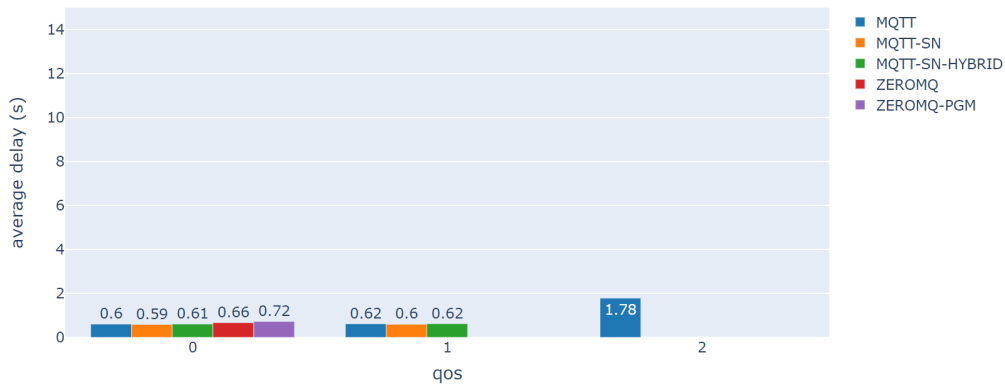


Figure 31: Average transmit delay times using the SATCOM network model with GPS data

Figure 31 shows how the protocols performed in regards to average transmit delay during the tests with the SATCOM network model using GPS data. Table 28 shows the packet loss for each protocol. As with the previous network models, we can see that the average delay is similar across the protocols. The only exception is MQTT with QoS 2, which has a delay of almost two seconds. This is again because QoS 2 introduces quite a bit of overhead in the form of handshakes and acknowledgments compared to QoS 0 and 1. The average delay is somewhat higher than the previous network models, but this was expected as the SATCOM network model introduces a delay of 550 milliseconds.

Protocol	QoS	Packet Loss
MQTT	0	0%
	1	0%
	2	0%
MQTT-SN Standard	0	0%
	1	0%
MQTT-SN Hybrid	0	0%
	1	0%
ZeroMQ TCP	None	0%
ZeroMQ PGM	None	0%

Table 28: Packet loss from tests with the SATCOM network model using GPS data

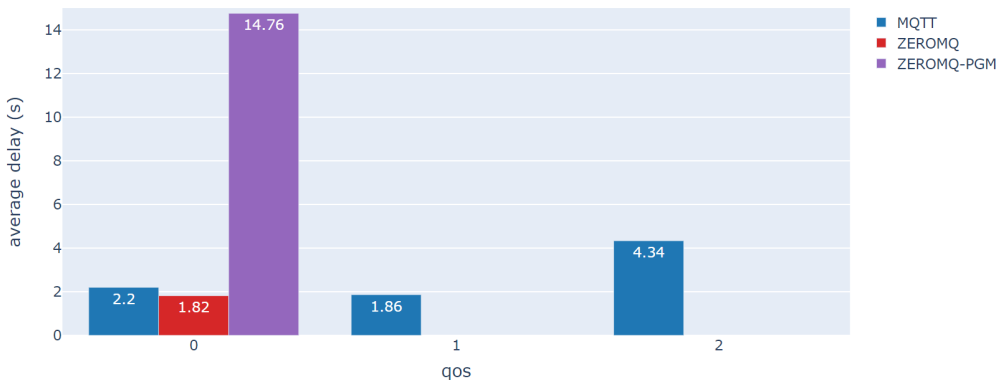


Figure 32: Average transmit delay times using the SATCOM network model with image data

Figure 32 shows how the protocols performed in regards to average transmit delay during the tests with the SATCOM network model using image data. Table 29 shows the packet loss for each protocol. The results follow a similar pattern to the other tests with image data in that the ZeroMQ using the PGM protocol is slower than MQTT. The main difference we see here is that ZeroMQ with TCP has a lower transmit delay than MQTT, even with QoS 0. In the previous comparison with Tactical Broadband, we saw that ZeroMQ had a much higher delay than MQTT. PGM also had a significant packet loss, which we have not experienced while testing any other protocol using SATCOM. There was also some packet loss from the Tactical Broadband test, but this was very little and may be due to connection issues at startup. The packet loss for SATCOM is much higher at over 12%, which can not be attributed to startup issues. The results also show that the delay has increased more for MQTT than for the tests using GPS data. This was especially the case with QoS 2. We know from the results on the network layer that the data rate decreased significantly in the test using this network model compared to the other models.

Protocol	QoS	Packet Loss
MQTT	0	0%
	1	0%
	2	0%
MQTT-SN Standard	0	0%
	1	0%
MQTT-SN Hybrid	0	0%
	1	0%
ZeroMQ TCP	None	0%
ZeroMQ PGM	None	12.77%

Table 29: Packet loss from tests with the SATCOM network model using image data

6.4 NATO Narrowband

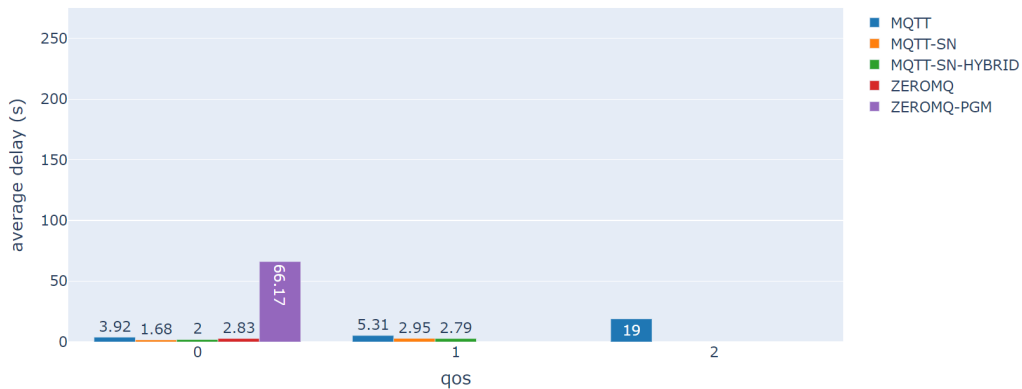


Figure 33: Transmit delay times using the NATO Narrowband network model with GPS data

Figure 33 shows how the protocols performed in regards to average transmit delay during the tests with the NATO Narrowband network model using GPS data. Table 30 shows the packet loss for each protocol. Here the results indicate that MQTT starts to struggle with higher levels of QoS. There is a slight increase in delay from QoS 0 to 1, and then a large increase in delay when QoS 2 is used. Both variations of the MQTT-SN implementation perform well even though the network has increased limitations. In this test, the standard variation of MQTT-SN outperformed the hybrid. ZeroMQ with TCP also showed lower delays than MQTT, even with QoS 0. Looking at the network statistics,

ZeroMQ had both a higher data rate and a lower amount of retransmissions and duplicate acknowledgements. ZeroMQ with PGM showed a high average delay, significantly higher than any of the other protocols, indicating that it struggles in this network. We can also see this from the packet loss and the fact that PGM is the only protocol with any messages lost.

Protocol	QoS	Packet Loss
MQTT	0	0%
	1	0%
	2	0%
MQTT-SN Standard	0	0%
	1	0%
MQTT-SN Hybrid	0	0%
	1	0%
ZeroMQ TCP	None	0%
ZeroMQ PGM	None	48.37%

Table 30: Packet loss from tests with the NATO Narrowband network model using GPS data

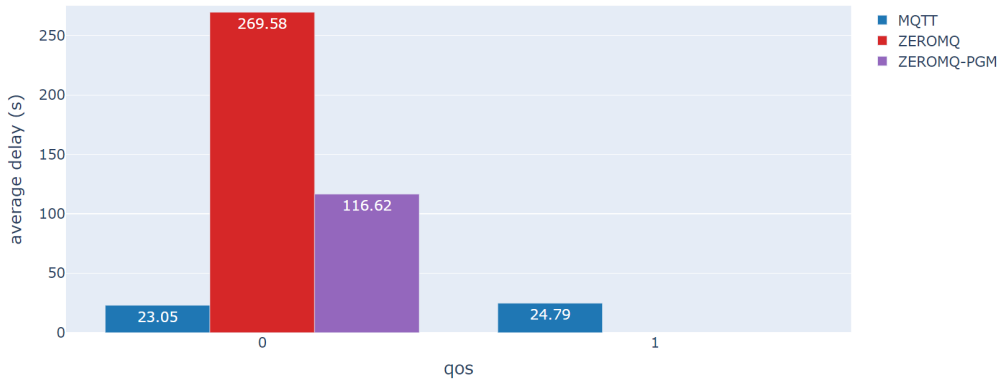


Figure 34: Average transmit delay times using the NATO Narrowband network model with image data

Figure 34 shows how the protocols performed in regards to average transmit delay during the tests with the NATO Narrowband network model using image data. Table 31 shows the packet loss for each protocol. Here we can observe that all protocols struggle to transmit messages efficiently regarding both the amount of delay and packet loss. MQTT has the lowest delay, but it is still high compared to the GPS test. ZeroMQ with TCP has the highest average delay. Looking at the network statistics, we can see that the message size is much larger than the previous network models. This indicates that TCP tries to merge messages into a single packet before sending it over the network. This might cause delays since TCP will spend time creating the new packets before trying to send them over the network. Most of these packets will fail to be sent, causing TCP to retry the transmission, causing a delay. This will also cause a bottleneck on the network, causing other packets not to be sent. We can also observe a large increase in retransmissions and duplicate acknowledgments compared to the other network models. From the network statistics for PGM, we see that the average message size was considerably smaller than with the previous network models, indicating that the protocol splits the message up into many multi-messages. The protocol tries to guarantee the delivery of all parts of a multi-part message, which causes very few messages to arrive, which can be seen from the amount of packet loss.

Protocol	QoS	Packet Loss
MQTT	0	85.86%
	1	91.09%
	2	100%
MQTT-SN Standard	0	NA
	1	NA
MQTT-SN Hybrid	0	NA
	1	NA
ZeroMQ TCP	None	76.56%
ZeroMQ PGM	None	89.48%

Table 31: Packet loss from tests with the NATO Narrowband network model using image data

6.5 CNR with 1% Loss

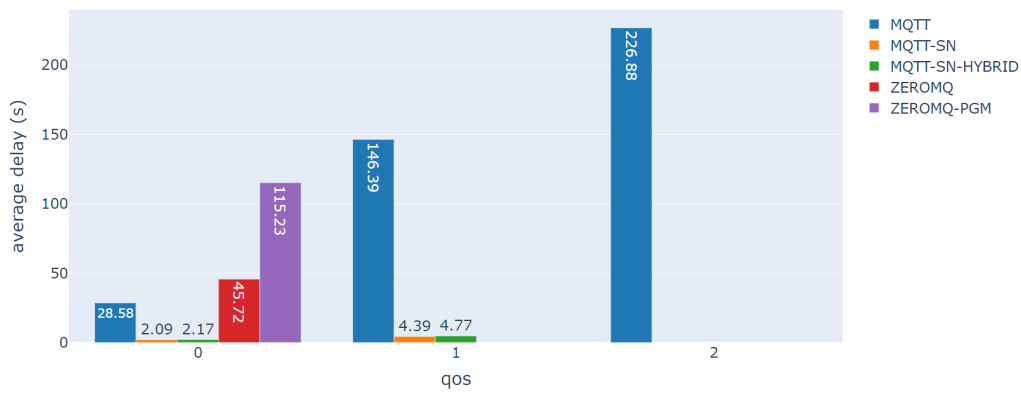


Figure 35: Average transmit delay times using the CNR network model with 1% loss with GPS data

Figure 35 shows how the protocols performed in regards to average transmit delay during the tests with the CNR network model with 1% loss using GPS data. Table 32 shows the packet loss for each protocol. These results show that most protocols start to struggle with efficient message transmission in this type of network. MQTT has high delays with QoS 0, which only increases with QoS 1 and 2. The number of retransmissions and duplicate acknowledgments were also much higher during this test for MQTT. ZeroMQ also has high delays for both TCP and PGM. This was expected as CNR is one of the most limited networks we have investigated. Protocols using TCP will be limited by the number of packets sent across the network due to the guarantees set by the protocol itself and the QoS level and because TCP attempts to resend messages repeatedly. This was also the case with the NATO Narrowband network model, but we see an increase in retransmissions and duplicate acknowledgments during these tests. MQTT-SN is the only protocol that performs well in this network, with significantly lower delay than other protocols. This was the case for both QoS 0 and 1.

Protocol	QoS	Packet Loss
MQTT	0	21.01%
	1	0%
	2	23.26%
MQTT-SN Standard	0	1.11%
	1	0.74%
MQTT-SN Hybrid	0	0.93%
	1	1.11%
ZeroMQ TCP	None	8.35%
ZeroMQ PGM	None	73.88%

Table 32: Packet loss from tests with the CNR network model with 1% loss using GPS data

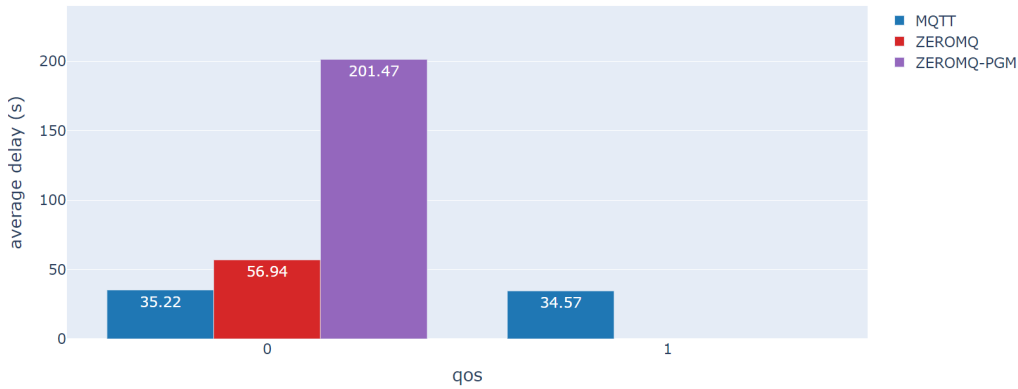


Figure 36: Average transmit delay times using the CNR network model with 1% loss with image data

Figure 36 shows how the protocols performed in regards to average transmit delay during the tests with the CNR network model with 1% loss using image data. Table 33 shows the packet loss for each protocol. Here we see similar results to that of the tests with NATO Narrowband, where all protocols have difficulties transmitting messages. MQTT still has the lowest average delay, but it is still high at 30 seconds. Here we see the lowest data rates compared to the other network models. For ZeroMQ, we also see the largest average message size, indicating that TCP attempts to package many messages into a single packet. As with MQTT, PGM had the lowest data rate during this test, and the average message size was also low from being split into multi-part messages.

Protocol	QoS	Packet Loss
MQTT	0	91.35%
	1	91.51%
	2	100%
MQTT-SN Standard	0	NA
	1	NA
MQTT-SN Hybrid	0	NA
	1	NA
ZeroMQ TCP	None	95.02%
ZeroMQ PGM	None	91.79%

Table 33: Packet loss from tests with the CNR network model with 1% loss using image data

6.6 CNR with 10% Loss

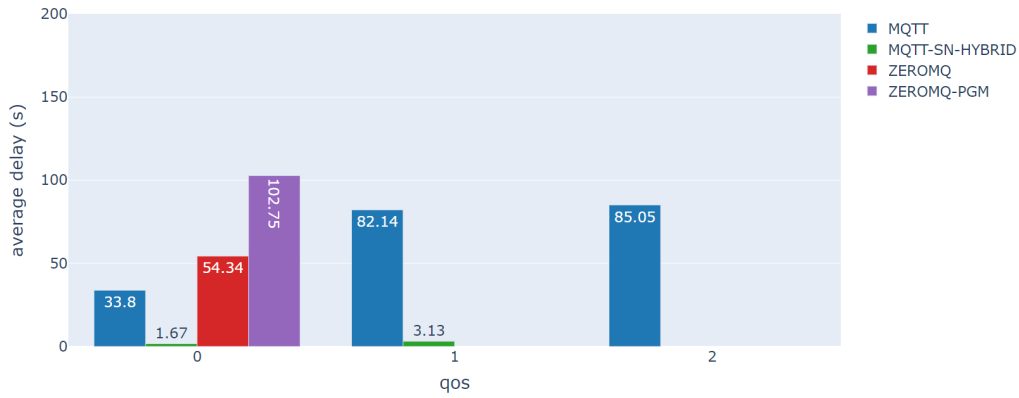


Figure 37: Average transmit delay times using the CNR network model with 10% loss with GPS data

Figure 37 shows how the protocols performed in regards to average transmit delay during the tests with the CNR network model with 10% loss using GPS data. Table 34 shows the packet loss for each protocol. This is the most constrained network that we have tested, and again we see that most of the protocols have high delays and can not deliver messages efficiently regarding speed. The exception is MQTT-SN using the hybrid test variation, which shows low delays. The standard variation of the MQTT-SN test did not manage to receive any messages and therefore performed worse than the hybrid variation. This is the only test showing a significant decrease in data rate while using MQTT-SN, which shows that this protocol struggles somewhat in transmitting messages.

Protocol	QoS	Packet Loss
MQTT	0	8.55%
	1	19.37%
	2	61.78%
MQTT-SN Standard	0	100%
	1	100%
MQTT-SN Hybrid	0	20.21%
	1	25%
ZeroMQ TCP	None	8.35%
ZeroMQ PGM	None	60.96%

Table 34: Packet loss from tests with the CNR network model with 10% loss using GPS data

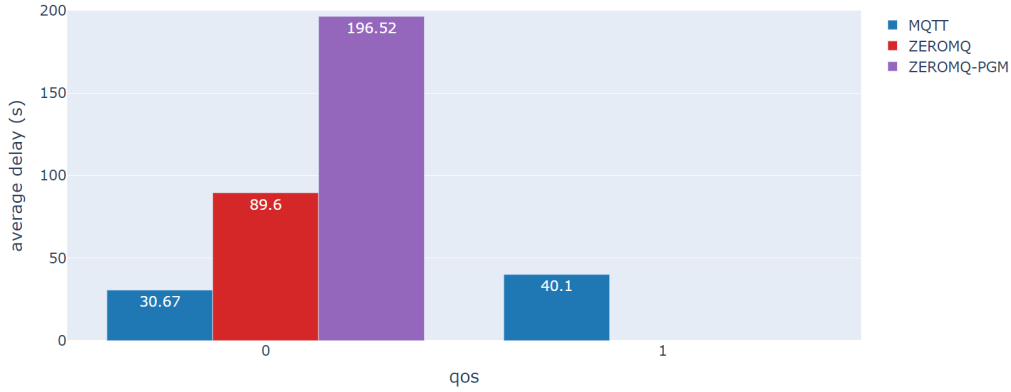


Figure 38: Average transmit delay times using the CNR network model with 10% loss with image data

Figure 36 shows how the protocols performed in regards to average transmit delay during the tests with the CNR network model with 1% loss using image data. Table 35 shows the packet loss for each protocol. We see similar results here where no protocols could transmit messages efficiently. This shows that these networks are challenging, even if the protocols are developed for use in limited networks with poor bandwidth.

Protocol	QoS	Packet Loss
MQTT	0	94.29%
	1	91.51%
	2	100%
MQTT-SN Standard	0	NA
	1	NA
MQTT-SN Hybrid	0	NA
	1	NA
ZeroMQ TCP	None	97.30%
ZeroMQ PGM	None	93.28%

Table 35: Packet loss from tests with the CNR network model with 10% loss using image data

6.7 MQTT

The MQTT protocol performs well in the least challenging network models, including 5G, Tactical Broadband and SATCOM. The average delay is relatively low across all tests compared to the other protocols, and there is no sign of any packet loss. Running with 27 clients that publish messages every 10 seconds does not seem to strain the protocol's transport capabilities with the configuration we have used in these tests. This implies that the MQTT protocol should be able to scale to even more clients or handle a higher data rate by lowering the time between published messages. Comparing the QoS in each of these tests, we can see that QoS 0 and 1 are comparable because they provide similar average delays. QoS 2, on the other hand, has a much higher average transmit time throughout the tests. Looking at network layer statistics, we can see that this is due to the increased overhead introduced by the reliability guarantees used by QoS 2.

The message sizes remain the same size for 5G, Tactical Broadband, SATCOM, and NATO Narrowband tests. The sizes increase drastically when using CNR with 1% and 10% loss compared to the other models. From observations done, this seems to happen due to messages being packaged together and sent as one message. This only applies to messages sent using QoS 1 and QoS 2. The reason for this might be due to TCP packet

batching, which is used to increase efficiency by reducing the number of packets sent over the network at a time.

Summarizing the MQTT protocol tests, we can see that the protocol performs well with the 5G, Tactical Broadband, and SATCOM network models. Running 27 clients of each type and a publishing delay of 10 seconds did not cause any significant performance issues, even with a larger message size. Both network models produce similar data rates, and there were very few retransmissions and duplicate acknowledgments relative to the number of packets sent.

While testing the NATO Narrowband network model, there were not many issues when sending GPS data. No messages were lost, and the data rate was similar to the previous models. The most significant change was the average delay, which was many times bigger than the previous models, but performance can still be considered reasonable. Using image data that produced larger message sizes, the test showed that the protocol struggled regardless of QoS level. Packet loss was high, and so was the delay in messages. The number of clients that disconnected was also very high.

The CNR network model tests with 1% and 10% loss had significant performance issues regardless of data type. Very few messages were sent compared to the tests with the other models, and packet loss was high. Sending larger messages with image data caused all packets to be lost for QoS 2 and nearly all to be lost using QoS 0 and 1. QoS 0 had the lowest delay during these tests but was still relatively high. Many clients disconnected during these tests, which can explain the low amount of messages sent. One discrepancy with the tests was that using image data and CNR with 1% loss, the number of disconnected clients was quite low compared to other tests.

6.8 MQTT-SN

Summarizing the results, we can see that the tests had good results using the 5G, Tactical Broadband, and SATCOM network models using both the standard and hybrid test variations. NATO Narrowband and CNR with 1% loss also produced good results while being a bit slower than the other models. The tests using the CNR network models with 1% loss showing good results can be somewhat surprising as it is one of the more limited models. These models also produced similar data rates, except 5G using the standard test variation.

The test using CNR with 10% showed poor results using the standard test, the data rate was very low, and the packet loss was 100%. The results from the hybrid test did show surprisingly good results, even if it did have some packet loss. Where all messages were lost using the standard version, the hybrid version had a maximum packet loss of 20.21% using QoS 0. The transmit delays were also low, being even lower than that of NATO Narrowband and CNR with a 1% loss.

The results will be affected by the changes we have made to the setup compared to MQTT. We are using fewer clients, which will have positively affected the results. Unfortunately, we have been limited in testing due to the clients we have used, which does not allow us to make a direct comparison to MQTT. The issues we have had with the clients may have affected the results from testing, which is important to consider.

6.9 ZeroMQ with TCP

Summarizing the results, we can see that ZeroMQ with TCP performs relatively well regarding reliability using the 5G, Tactical Broadband, SATCOM, and NATO Narrowband network models. Using GPS data, they all have similar data rates and average message sizes. The number of retransmissions and duplicate acknowledgments is also

very low. The delay found in the tests shows that the protocol is relatively slow with the test setup we have used. Tactical Broadband also had higher delays than what was expected. NATO Narrowband had very good results, looking at packet loss and delay. Compared to the less limited network models, it did have some TCP retransmissions and duplicate acknowledgments.

Using image data, 5G, Tactical Broadband, and SATCOM provided good results when looking at reliability. Fewer messages were sent, but there was zero packet loss recorded. NATO Narrowband showed poor results using image data. Most of the messages were lost, and the transmit delays were high. The data rate was also low, and the average message size increased a lot compared to the 5G, Tactical Broadband, and SATCOM models. CNR with 1% and 10% loss had even worse results, which shows that the increase in message size is not sustainable for these network models. We see that CNR with 10% loss did have a lower average delay than NATO Narrowband, but it should be noted that very few messages arrived.

6.10 ZeroMQ with PGM

Summarizing the results from ZeroMQ with PGM showed that the protocol performed well with the 5G, Tactical Broadband, and SATCOM network models using GPS data regarding the delay. The protocol struggled with larger message sizes and showed poor performance most tests. We see similar problems with the TCP transport protocol and there might be an issue with the configuration used for both of the protocols. We also use the same message broker, which might affect the results we are seeing. From the statistics on the network layer, we see that the protocol has a much larger data rate than any of the other protocols we have tested. This is due to the fact that this is the only multicast protocol used, whereas all the others are point-to-point (unicast). Multicast will create more packets than unicast because ZeroMQ packets are sent out to all possible clients on the network. The results showed that the data rate was much higher with image data than with GPS data. The results in regards to packet loss showed that several packets were lost, even using the Tactical Broadband and SATCOM network models. This was the only protocol that had packet loss during tests with these network models, which signifies issues. One aspect that could be looked more into is removing the message broker, which might have caused a bottleneck while using multicasting. There might also be certain configuration parameters that can be configured to make the protocol more efficient, although we tried several different configurations that had no effect.

6.11 Summary

The results show that most protocols achieve good performance in the less challenging networks like 5G, Tactical Broadband and SATCOM. The fact that the protocols perform well in a low-band 5G network is not surprising, as this network is the least demanding of the tested models. There are still some results that are a bit worse than expected. ZeroMQ with TCP, in particular, has relatively high delays compared to MQTT and MQTT-SN in some of the networks. The delays are still low, below one second, but this is much higher than the other protocols with only a few milliseconds of delay. The PGM protocol also has much higher transmit delays than TCP, which should not be the case due to less congestion on the network. This might be due to issues with the configuration of the protocol and problems getting multicasting to work with the virtual machine. Tuning the options should be considered for future studies.

It should be noted that the data rates captured during testing are reduced slightly because of the cool-down period after the initial duration of the tests. This was due to smaller ping request/reply packets sent because of the 60 seconds keep alive set when configuring the clients, especially with MQTT and MQTT-SN.

ZeroMQ also showed worse results for the Tactical Broadband and SATCOM network models using image data. While still not poor, the results were not as good as expected, considering the networks' relatively low challenge. It should be noted that during our testing with PGM, we ran into an issue with the firewall blocking our ability to send multicast messages, which had to be manually overridden to continue the sending of messages. This will have had some effect on the results and the delays found. While ZeroMQ supports PGM, it is not commonly used, and the implementation may not be as optimized as when running ZeroMQ with TCP. Compared to MQTT-SN, which uses UDP, it showed much better results overall than that of ZeroMQ with PGM. This may be because of the lower amount of clients used and the fact that we did not run with QoS 2 simultaneously, which would have produced more traffic on the network.

None of the protocols seemed to be able to handle the most challenging of the network models, except for MQTT-SN. Both the standard and the hybrid test variation handled CNR with a 1% loss reasonably well, and this was without much packet loss or delay. The hybrid test variation also seemed to handle CNR with 10% loss very well compared to most of the other protocols tested.

7 Recommendations

Based on the results from testing, we have decided on recommendations for each protocol to answer RQ2. The network models and their limitations have been considered when providing insight into which protocols should be used for a specific network. The evaluation criteria we have used to give recommendations are mainly based on the packet loss and latency experienced during testing. These metrics are compared to the inherent limitations set by each model, as can be seen from the network models in table 1. The expectations we have for each protocol will change based on how challenging the network is, but it should maintain the ability to send and receive messages efficiently.

7.1 Cutoffs

The following list explains the reasoning for the cutoffs we have set for recommending the protocol for use in the specified network model:

- For 5G, we have set the cutoff to be 200 milliseconds for the delay and 0% for packet loss. The 5G network model has an inherent delay of 20 milliseconds, which is relatively low. The cutoff has been set ten times higher as we know the protocol will add additional delay during transmission. We do not expect any packet loss from this network model because it does not simulate any loss, and it is the least limited network we are using.
- For Tactical Broadband, we have set the cutoff to be 400 milliseconds for the delay and 0% for packet loss. Compared to the 5G model, the data rate is much lower but still higher than many other network models. We expect an increase in delays compared to 5G due to the inherent delay of 100 milliseconds, but it should not be much higher than 5G. The model simulates a loss of 1%, but we do not expect any packets to be lost due to the TCP protocol, as TCP will try to resend them if they fail.
- For SATCOM, we have set the cutoff to be 750 milliseconds for the delay and 0% for packet loss. For image data, we have increased the cutoff to 2.5 seconds for delay. SATCOM has a lower data rate than 5G and Tactical Broadband, but it should still be sufficient for sending smaller messages without any issues. The inherent delay is somewhat higher than the other network models we have used, at 550 milliseconds. However, we do not expect much additional delay time other than what the protocol adds. Therefore, we also do not expect any packet loss from this model. For image data, we have increased the cutoff to 2.5 seconds due to the lower data rate, which can affect sending larger messages.
- For NATO Narrowband, we have set the cutoff to be 4 seconds for the delay and 5% for packet loss. For image data, the cutoff for the delay is increased to 10 seconds, and the packet loss should be less than 10%. The NATO Narrowband network model is one of the more challenging models we have tested, and therefore the requirement for delay and packet loss is lower. The protocols should still be required to transmit messages in a manner considered efficient. Our cutoff for the delay is quite a bit higher than the previous models, which is mainly due to the low data rate, which will affect the transmission of messages, especially with TCP, as the protocol will cause a large amount of traffic on the network as it tries to resend dropped packets. For this reason, we have increased the cutoff for image data, as this will cause even larger packets to travel over the network. We also expect some packet loss due to congestion on the network. The cutoff for delay and packet loss for image data should not be higher, as it would not meet the requirements for fast and efficient communication.
- For both CNR models, we have set the cutoff to be 10 seconds for delay and 10% for packet loss. The CNR models we have tested are the most challenging of the

network models we have looked at and also have the highest overall cutoffs. The requirements are the same for both GPS and image data, as any higher delay than this will be too slow, and any higher packet loss will be too unreliable.

7.2 MQTT

Table 36 shows the recommendations for MQTT. We have provided one recommendation for each network model and one for each of the two data types we have used. If the protocol is recommended, there is also a recommendation for the QoS level. We have recommended MQTT for use in the 5G, Tactical Broadband, and the SATCOM network models. The protocol showed good results for both data types, meaning that it can handle a range of message sizes. No significant extra delay was introduced with the protocol on top of the limitations from each network model. We only recommend using QoS 0 and QoS 1, as these provided the most efficient transmission. QoS 2 has too much overhead, and the additional benefits are not worth the additional delay added in any of the networks. MQTT is not recommended for use in any more challenging networks, except for NATO Narrowband with GPS data. The protocol can handle this network model well with smaller message sizes. We only recommend using QoS 0 with this network model, as QoS 1 introduces too much additional delay.

Protocol	Network Model	Data Type	Recommended	QoS
MQTT	5G	GPS Data	Yes	0 and 1
		Image Data	Yes	0 and 1
	Tactical Broadband	GPS Data	Yes	0 and 1
		Image Data	Yes	0 and 1
	SATCOM	GPS Data	Yes	0 and 1
		Image Data	Yes	0 and 1
	NATO Narrowband	GPS Data	Yes	0
		Image Data	No	
	CNR 1% Loss	GPS Data	No	
		Image Data	No	
	CNR 10% Loss	GPS Data	No	
		Image Data	No	

Table 36: Recommendations for MQTT

7.3 MQTT-SN

Table 37 shows the recommendations for MQTT-SN. The table is divided up into the recommendations for the standard variation and the hybrid variation. Both variations showed similar results for the different network models. MQTT-SN is recommended for use in the 5G, Tactical Broadband, SATCOM, NATO Narrowband, and CNR with 1% loss network models. It showed good performance in all of these networks using smaller message sizes. We only tested with QoS 0 and 1, and we recommend using both in all networks except CNR. CNR with 10% loss showed too much packet loss and is therefore not recommended.

Protocol	Variation	Network Model	Recommended	QoS
MQTT-SN	Standard	5G	Yes	0 and 1
		Tactical Broadband	Yes	0 and 1
		SATCOM	Yes	0 and 1
		NATO Narrowband	Yes	0 and 1
		CNR 1% Loss	Yes	0
		CNR 10% Loss	No	
	Hybrid	5G	Yes	0 and 1
		Tactical Broadband	Yes	0 and 1
		SATCOM	Yes	0 and 1
		NATO Narrowband	Yes	0 and 1
		CNR 1% Loss	Yes	0
		CNR 10% Loss	No	

Table 37: Recommendations for MQTT-SN

7.4 ZeroMQ with TCP

Table 38 shows the recommendations for ZeroMQ using the TCP protocol for transport. The protocol works well with the 5G, Tactical Broadband, SATCOM, and NATO Narrowband network models, but only with smaller message sizes. The protocol did not show good results with a larger message size and is therefore not recommended for use, unlike MQTT and MQTT-SN. The exception is with the SATCOM network model, where the protocol showed good results. The results were even better than we experienced with MQTT using QoS 0. The protocol struggles too much in the CNR network models to be recommended for usage in this type of network. In CNR with 1% loss, this is mostly due to the delay, as the packet loss is relatively low.

Protocol	Network Model	Data Type	Recommended
ZeroMQ (TCP)	5G	GPS Data	Yes
		Image Data	No
	Tactical Broadband	GPS Data	Yes
		Image Data	No
	SATCOM	GPS Data	Yes
		Image Data	Yes
	NATO Narrowband	GPS Data	Yes
		Image Data	No
	CNR 1% Loss	GPS Data	No
		Image Data	No
	CNR 10% Loss	GPS Data	No
		Image Data	No

Table 38: Recommendations for ZeroMQ with TCP

7.5 ZeroMQ with PGM

Table 39 show the recommendations for ZeroMQ using PGM as the transport protocol. Here the recommendations are similar to that of ZeroMQ with TCP. The exception is NATO Narrowband and SATCOM with image data which showed very high delays compared to the other protocols, making it unsuitable in this type of network. PGM showed slightly worse results than the other protocols in most networks, but we still recommend it based on our evaluation criteria. The recommendations are only for smaller message sizes, which is expected as it is a multicast protocol.

Protocol	Network Model	Data Type	Recommended
ZeroMQ (PGM)	5G	GPS Data	Yes
		Image Data	No
	Tactical Broadband	GPS Data	Yes
		Image Data	No
	SATCOM	GPS Data	Yes
		Image Data	No
	NATO Narrowband	GPS Data	No
		Image Data	No
	CNR 1% Loss	GPS Data	No
		Image Data	No
	CNR 10% Loss	GPS Data	No
		Image Data	No

Table 39: Recommendations for ZeroMQ with PGM

8 Conclusion

In this thesis, we have investigated several protocols that fall within the publish/subscribe pattern used for message transportation. We have looked at how these protocols perform in various military network models considered DIL. The research has been conducted to try and answer the following research questions:

1. **RQ1:** Can we correctly evaluate the performance of publish/subscribe protocols in DIL networks?
2. **RQ2:** Which DIL networks are the publish/subscribe protocols most suited for?

To answer RQ1, we can say ‘Yes’ to the fact that we can correctly evaluate the performance of publish/subscribe protocols in DIL networks. The tool we have created to evaluate the protocols bases its emulation on Netem, which is proven to be correct (see chapter 3, chapter 4 and chapter 5). The emulation of the DIL networks has been validated using iPerf3, which shows that the parameters we have used to limit the network have been correctly applied. Our findings on both the network and application layer show that the results correlate to that of previous studies with the Tactical Broadband network model [12].

In order to evaluate these protocols, we have run several tests using different data types to see how the protocols performed in these networks. The main metrics we have looked at are delay and packet loss, found by statistics taken on the application layer. We have also looked at other metrics related to the network layer. In order to run these tests, we have developed an analysis tool that is built explicitly for this purpose.

To answer RQ2, we can from our results in chapter 5, the analysis in chapter 6, and our recommendations in chapter 7 see that most protocols are suited for use in less challenging networks like 5G, Tactical Broadband and SATCOM. For MQTT, this is the case with both small and large message sizes. For ZeroMQ, this is not the case, and we find that the protocol is mostly suitable with smaller message sizes, regardless of whether the transport protocol is TCP or PGM. We can see that the MQTT-SN protocol is most suited for use in DIL networks. This answer is more complicated since the protocol is not evaluated on the same premises as the other protocols. Still, the results were good, and the increase in clients should not have that big of an impact on the results in regards to delay and packet loss. Since MQTT-SN uses UDP, it does not have the additional guarantees that come with TCP and we can see that using it has been beneficial in the more limited networks we have used.

Based on our analysis, we have come to a conclusion for each of the protocols based on how they performed in each of the networks.

8.1 MQTT

MQTT shows good performance in the less limiting networks we have tested. The protocol shows no issues in transmitting messages, both in regards to delay and packet loss. Using QoS 2 showed that the amount of overhead caused issues for the protocol, primarily due to the extra steps needed before a message transmission was completed. Due to this, we only recommend using QoS 0 and 1, as the overhead is mostly reduced with these settings. The tests with more limiting networks show that the protocol struggled and is therefore not found suitable for use. The protocol was the easiest to set up and use compared to the other protocols, and this is to be expected, as MQTT is an industry-standard protocol in IoT.

8.2 MQTT-SN

Our results show that MQTT-SN had good results for almost all tests. The delay was low with both test variations across all network models, and the protocol outperformed the other ones in most tests. The packet loss was also comparatively low. Both QoS 0 and 1 showed good results, but QoS 0 was most times faster than QoS 1. The only issues were found during the test using the CNR network model with a 10% loss. Here, only the hybrid test variation could send and receive messages. Even though the network is quite limited, it showed good results, both in terms of delay and packet loss. Still, the packet loss was too high for us to be able to recommend its usage in such a network. While overall results are good, the implementation limitations have restricted us from adequately being able to compare it to the other protocols. Setting up the protocol was not an easy task and was significantly more difficult than the other two protocols we have investigated, except for ZeroMQ with PGM.

8.3 ZeroMQ

ZeroMQ showed the overall poorest results from the protocols we have tested. TCP had some promise with smaller message sizes, where the results were comparable to that of MQTT and MQTT-SN. However, the test with larger message sizes showed that the protocol had issues, both in terms of delay and packet loss. PGM provided the overall worst results, especially with larger message sizes. The poor results might be due to the test implementation, which is difficult to say without more testing. The results may have been better without a broker, but the downside is the increased complexity in setting up and connecting the clients. Setting up the protocol with clients was relatively simple, but there were certain networking issues with getting PGM to work as it is not well supported in ZeroMQ.

9 Future Work

After the studies that have been conducted in this thesis, we have some topics that we would like to propose for future studies into evaluating publish/subscribe protocol in challenging networks.

9.1 Developing an MQTT-SN Client

Our results show good promise for the use of MQTT-SN. The issue with our testing is that we could not create a test environment comparable to what was used for MQTT and ZeroMQ. This was primarily due to difficulties finding a Python client that we could incorporate into our analysis tool, and we had to use a custom client with several limitations. By creating our client from the ground up, we can better understand the client and make it work better with the test requirement we have to evaluate the protocol. The benefits can also be found in allowing us more ways to understand events that happen during a test, e.g., implementing a callback feature for when a client disconnects. This would also allow us to test with QoS 2 and larger message sizes, as we can then tailor the client to the MQTT-SN specifications correctly.

9.2 Additional Testing with ZeroMQ

During this thesis, we have evaluated ZeroMQ with both the TCP and PGM protocols for message transport. ZeroMQ supports several protocols for this purpose, and it would be interesting to investigate how it performs using UDP. Since MQTT-SN showed promising results while using UDP, it would be valuable to also gather data on how it compares to ZeroMQ. It must be said that there was little support for PGM in ZeroMQ, and this might also be the case with UDP, but it should still be tested as the implementation should not be considerably difficult.

ZeroMQ provides a range of configuration options, and we have only used a basic configuration for this thesis. The performance could be improved with some changes to the setup, which would be interesting to experiment with. A solution could be to create a setup that did not use a message broker, as this could affect the performance by becoming a bottleneck.

9.3 Configuring the TCP Stack

The TCP stack in the Linux kernel is highly customizable and could therefore be tweaked to gain performance benefits while using the publish/subscribe protocols we have used in this thesis. A user-level TCP stack can potentially remove some of the Linux TCP overhead by eliminating system calls [65].

9.4 Open Sourcing the Analysis Tool

The tool we have created during the preparatory project and further developed in this thesis has allowed us to conduct tests on various publish/subscribe protocols in a manner that was repeatable, controlled, and reliable. The tool also allows for a dynamic configuration, meaning that the user can perform tests that differ from the ones we have conducted and tailor them to suit their test requirements. By making it open source, we allow other researchers and developers to use the tool for purposes of testing protocols, as well as modify it to their own needs, e.g., implementing support for additional protocols.

References

- [1] Gartner, *Internet of things (iot)*, <https://www.gartner.com/en/information-technology/glossary/internet-of-things>, Accessed on 2022.06.08.
- [2] Matthew O’Riordan, *Everything You Need To Know About Publish/Subscribe*, <https://ably.com/topic/pub-sub>, Accessed on 2022.06.07.
- [3] OASIS, *MQTT Version 3.1.1*, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718018, Accessed on 2022.06.08.
- [4] —, *MQTT For Sensor Networks (MQTT-SN) Protocol Specification Version 1.2*, https://www.oasis-open.org/committees/download.php/66091/MQTT-SN_spec_v1.2.pdf, Accessed on 2022.06.08.
- [5] The ZeroMQ authors, *ZeroMQ An open-source universal messaging library*, <https://zeromq.org/>, Accessed on 2022.06.08.
- [6] K. L. Scott, T. Refaei, N. Trivedi, J. Trinh and J. P. Macker, ‘Robust communications for disconnected, intermittent, low-bandwidth (DIL) environments’, *2011 - MILCOM 2011 Military Communications Conference*, pp. 1009–1014, 2011.
- [7] Claes Wohlin, Martin Höst, Kennet Henningsson, *13 Empirical Research Methods in Web and Software Engineering*, <http://www.wohlin.eu/web05.pdf>, Accessed on 2022.06.08.
- [8] Emil P. Andersen, *Creating an analysis tool for testing and evaluating the mqtt protocol*, https://github.com/EPA1/Preparatory-Project/blob/main/NTNU_Preparatory-Project.pdf, Accessed on 2022.06.08.
- [9] T. H. Bloebaum, N. J. Kevin Chan, F. T. Johnsen and A. T. Marco Manso, ‘Nato core services profiling for hybrid tactical networks’, NORTH ATLANTIC TREATY ORGANIZATION, AC/323(IST-150)TP/1008, Mar. 2021. [Online]. Available: [https://www.sto.nato.int/publications/STO%5C%20Technical%5C%20Reports/STO-TR-IST-150/%5C%5C\\$TR-IST-150-ALL.pdf](https://www.sto.nato.int/publications/STO%5C%20Technical%5C%20Reports/STO-TR-IST-150/%5C%5C$TR-IST-150-ALL.pdf).
- [10] M. Manso, K. C. Norman Jansen, T. H. B. Andrew Toth and F. T. Johnsen, *Mobile Tactical Force Situational Awareness: Evaluation of Message Broker Middleware for Information Exchange*, ICCRTS 2018, Pensacola, Florida, USA.
- [11] Z. Kang and A. Dubey, ‘Evaluating DDS, MQTT, and ZeroMQ Under Different IoT Traffic Conditions’, 2020. [Online]. Available: <http://www.dre.vanderbilt.edu/~gokhale/WWW/papers/M4IoT2020>.
- [12] F. Johnsen *et al.*, ‘Evaluating publish/subscribe standards for situational awareness using realistic radio models and emulated testbed’, Oct. 2019.
- [13] Z. Shelby, K. Hartke and C. Bormann, *The Constrained Application Protocol (CoAP)*, RFC 7252, Jun. 2014. DOI: 10.17487/RFC7252. [Online]. Available: <https://www.rfc-editor.org/info/rfc7252>.
- [14] Carsten Bormann, *Coap*, <https://coap.technology/>, Accessed on 2022.06.08.
- [15] David Pike, *The importance of communications in the military*, <https://blog.samtec.com/post/the-importance-of-communications-in-the-military/>, Accessed on 2022.06.08, Aug. 2021.
- [16] NATO IST-124 Research Task Group, *Anglova Scenario*, <https://anglova.net/>, Accessed on 09.06.2022.
- [17] N. Suri *et al.*, ‘The angloval tactical military scenario and experimentation environment’, in *2018 International Conference on Military Communications and Information Systems (ICMCIS)*, 2018, pp. 1–8. DOI: 10.1109/ICMCIS.2018.8398729.
- [18] M. Pradhan, ‘Interoperability for disaster relief operations in smart city environments’, in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, 2019, pp. 711–714. DOI: 10.1109/WF-IoT.2019.8767169.

-
- [19] European Commission website, *Smart cities*, https://ec.europa.eu/info/eu-regional-and-urban-development/topics/cities-and-urban-development/city-initiatives/smart-cities_en, Accessed on 2022.06.07.
- [20] Steven Walker, Daniel Rice, Mark Kahn, John Clark, *Why the World's Militaries are embracing 5G*, <https://spectrum.ieee.org/lockheed-martin-5g>, Accessed on 2022.06.08.
- [21] ASELSAN, *GRC-5220 Tactical Broadband ETHERNET Radio*, <https://www.aselsan.com.tr/en/capabilities/military-communication-systems/military-broadband-multimode-radiolinks/grc5220-tactical-broadband-ethernet-radio>, Accessed on 2022.06.08.
- [22] NATO, *SATCOM Post-2000 (Archived)*, https://www.nato.int/cps/en/natohq/topics_50092.htm, Accessed on 2022.06.08.
- [23] Tore J. Berg, *NATO Narrowband Waveform (NBWF) - multilevel-level precedence and preemption for IP traffic*, <https://ffi-publikasjoner.archive.knowledgearc.net/handle/20.500.12242/953>, Accessed on 2022.06.08.
- [24] military-history.fandom, *Combat-net radio*, https://military-history.fandom.com/wiki/Combat-net_radio, Accessed on 2022.06.08.
- [25] U.S. Naval War College, *Humanitarian Assistance and Disaster Response (HA/DR): Home*, <https://usnwc.libguides.com/hadr>, Accessed on 2022.06.08.
- [26] Randall R. Stewart, *Stream Control Transmission Protocol*, RFC 4960, Sep. 2007. DOI: 10.17487/RFC4960. [Online]. Available: <https://rfc-editor.org/rfc/rfc4960.txt>.
- [27] IBM, *MQTT restrictions and limitations*, <https://www.ibm.com/docs/en/watson-iot-platform?topic=messaging-restrictions-limitations>, Accessed on 2022.06.07.
- [28] Steve Cope, *Introduction to MQTT-SN (MQTT for Sensor Networks)*, <http://www.steves-internet-guide.com/mqtt-sn/>, Accessed on 2022.06.08.
- [29] Packt, *Understanding wildcards*, <https://subscription.packtpub.com/book/application-development/9781787287815/1/ch01lv1sec18/understanding-wildcards>, Accessed on 2022.06.08.
- [30] The ZeroMQ authors, *Chapter 1 - Basics*, <https://zguide.zeromq.org/docs/chapter1/>, Accessed on 2022.06.08.
- [31] Tony Speakman et al., *PGM Reliable Transport Protocol Specification*, RFC 3208, Dec. 2001. DOI: 10.17487/RFC3208. [Online]. Available: <https://www.rfc-editor.org/info/rfc3208>.
- [32] O.S Tezer, *How To Work with the ZeroMQ Messaging Library*, <https://www.digitalocean.com/community/tutorials/how-to-work-with-the-zeromq-messaging-library>, Accessed on 2022.06.08.
- [33] Andreas Giannopoulos, *D58: Pragmatic General Multicast*, http://www0.cs.ucl.ac.uk/research/darpa/pgm/PGM_FINAL.html, Accessed on 2022.06.08.
- [34] The ZeroMQ authors, *Chapter 2 - Sockets and Patterns*, <https://zguide.zeromq.org/docs/chapter2/>, Accessed on 2022.06.08.
- [35] Martin Sustrik, Martin Lucina, *Zmq-setsockopt(3)*, <http://api.zeromq.org/2-1:zmq-setsockopt>.
- [36] Object Management Group, Inc, *What is DDS?*, <https://www.dds-foundation.org/what-is-dds-3/>, Accessed on 2022.06.08.
- [37] N. Suri et al., 'Evaluating the scalability of group communication protocols over synchronized cooperative broadcast', in *2021 International Conference on Military Communication and Information Systems (ICMCIS)*, 2021, pp. 1–9. DOI: 10.1109/ICMCIS52405.2021.9486407.
- [38] NATS Doc, *The Streaming Protocol*, <https://docs.nats.io/legacy/stan/streaming/protocol>, Accessed on 2022.06.08.
- [39] N. Suri et al., 'Diservice: A peer-to-peer disruption tolerant dissemination service', Oct. 2009, pp. 1–8, ISBN: 1424452384. DOI: 10.1109/MILCOM.2009.5379952.
-

-
- [40] The ZeroMQ authors, *Norm transport notes*, <http://wiki.zeromq.org/topics:norm-protocol-transport>, Accessed on 2022.06.08.
- [41] The Tcpdump Group, *Tcpdump and libcap*, <https://tcpdump.org/>, Accessed on 2022.06.08.
- [42] G. Harris, Ed. and M. Richardson Sandelman, *Pcap capture file format*, <https://tools.ietf.org/id/draft-gharris-opsawg-pcap-00.html>, Accessed on 2022.06.08.
- [43] Wireshark Foundation, *About Wireshark*, <https://www.wireshark.org/>, Accessed on 2022.06.08.
- [44] Matthew Treinish and Jeremy Stanley, *Pymqttbench*, <https://github.com/mtreinish/pymqttbench>, Accessed on 2022.06.08.
- [45] J. Ahrenholz, T. Goff and B. Adamson, ‘Integration of the core and emane network emulators’, in *2011 - MILCOM 2011 Military Communications Conference*, 2011. DOI: 10.1109/MILCOM.2011.6127585.
- [46] A. Jurgelionis, J.-P. Laulajainen, M. Hirvonen and A. I. Wang, ‘An empirical study of netem network emulation functionalities’, in *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, 2011, pp. 1–6. DOI: 10.1109/ICCCN.2011.6005933.
- [47] NumFOCUS, *Pandas*, <https://pandas.pydata.org/>, Accessed on 2022.06.08.
- [48] Plotly, *Plotly dash*, <https://plotly.com/dash/>, Accessed on 2022.06.08.
- [49] Oracle Corporation, *MySQL*, <https://www.mysql.com/>, Accessed on 2022.06.08.
- [50] Linux Foundation, *Netem*, <https://wiki.linuxfoundation.org/networking/netem>, Accessed on 2022.06.08.
- [51] OpenStreetMap contributors, *Dash Leaflet*, <https://dash-leaflet.herokuapp.com/>, Accessed on 2022.06.08.
- [52] Oracle Corporation, *MySQL Connector/Python Developer Guide*, <https://dev.mysql.com/doc/connector-python/en/>, Accessed on 2022.06.08.
- [53] —, *Virtualbox*, <https://www.virtualbox.org/>, Accessed on 2022.06.08.
- [54] National Air and Space Museum, *The Blue Force Tracker System*, <https://timeandnavigation.si.edu/multimedia-asset/the-blue-force-tracker-system>, Accessed on 2022.06.08.
- [55] J. Dugan et al., *iPerf - The ultimate speed test tool for TCP, UDP and SCTP*, <https://iperf.fr/>, Accessed on 2022.06.08.
- [56] Eclipse Foundation, *Eclipse Mosquitto™ An open source MQTT broker*, <https://mosquitto.org/>, Accessed on 2022.06.08.
- [57] —, *Paho-mqtt*, <https://pypi.org/project/paho-mqtt/>, Accessed on 2022.06.08.
- [58] —, *MQTT-SN Transparent Gateway*, <https://eclipse.org/paho/index.php?page=components/mqtt-sn-transparent-gateway/index.php>, Accessed on 2022.06.08.
- [59] Steve Cope, *Using The Python MQTT-SN Client*, <http://www.steves-internet-guide.com/python-mqttsn-client/>, Accessed on 2022.06.08.
- [60] The ZeroMQ authors, *Chapter 5 - Advanced Pub-Sub Patterns*, <https://zguide.zeromq.org/docs/chapter5/>, Accessed on 2022.06.08.
- [61] rickyten, *Zeromq xpub xsub example.md*, <https://gist.github.com/rickyten/0acfc9f42325e59ff6360a9a82035e50>, Accessed on 2022.06.08.
- [62] Linux Manual Page, *Raw(7)*, <https://man7.org/linux/man-pages/man7/raw.7.html>, Accessed on 2022.06.08.
- [63] T. Bray, ‘The JavaScript Object Notation (JSON) Data Interchange Format’, Google, Inc., RFC 7159, Mar. 2014, pp. 1–15. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc7159>.
- [64] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub and S. Hagen, *The GeoJSON Format*, RFC 7946, Aug. 2016. DOI: 10.17487/RFC7946. [Online]. Available: <https://rfc-editor.org/rfc/rfc7946.txt>.
-

-
- [65] E. Y. Jeong *et al.*, ‘Mtcp: A highly scalable user-level tcp stack for multicore systems’, in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI’14, Seattle, WA: USENIX Association, 2014, pp. 489–502, ISBN: 9781931971096.

Appendix

A Additional Details About the Analysis Tool

The analysis tool includes the test framework, frontend, and database connector modules as specified in Chapter 3. The project includes a `requirements.txt` file which will install the required dependencies for the project. Installing the requirements can be achieved through the command `pip install -r requirements.txt -v`. Once the requirements are installed, the program can be started by running the `main.py` file, located at the root of the project directory.

Data used for packaging messages are found in the `data` folder in the root directory. We have both GPS data and image data. The GPS data comes from several GPX files containing coordinates (latitude and longitude). The image data is a single file used for all tests. Any data can be added or removed from these folders. For GPS data, the file name format must be specified as a node's name, e.g., `NOR-UNIT001.gpx`.

Due to time constraints, we could not make the frontend fully configurable, meaning that some settings are stored and loaded through JSON files. These files can be found in the `config` folder in the root directory. These include some settings that are already handled by the frontend, such as protocol and test duration, while others relate to the name of the nodes, their topic, and the IP address of the broker.

Data saved locally from each test are stored in the `logs` folder, which is found at the project's root. Logs are divided up into live and post data, where live is data from every publisher node and every subscriber node. Post data are collected data and statistics. This is also where the network data is saved in a PCAP file.

The database is tightly coupled to the tool and is located on an NTNU server. As mentioned in Chapter 3, the connection to the database requires the user to either be on an NTNU network or be connected through one with the use of VPN. Unfortunately, this requirement is a limitation in the tool's usage, as the user cannot fetch or upload data without a connection to the database. In chapter 9, section 9.4 we mention that the tool should be open sourced in the future. This would include removing the dependency of having the database on an NTNU server and allowing custom data storage solutions.

B Virtual Machine Configuration

This section describes the configuration of the virtual machine that we have used. Most of the settings we have used are the default ones when configuring a virtual machine through VirtualBox.

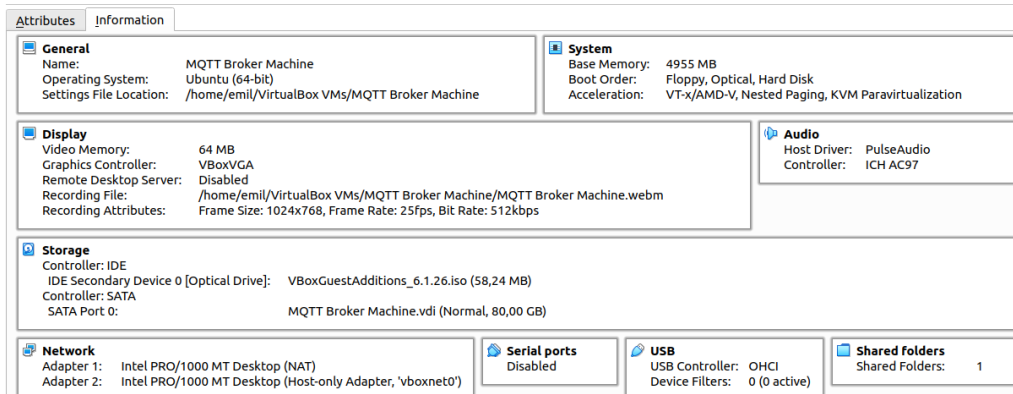


Figure 39: Virtual Machine settings

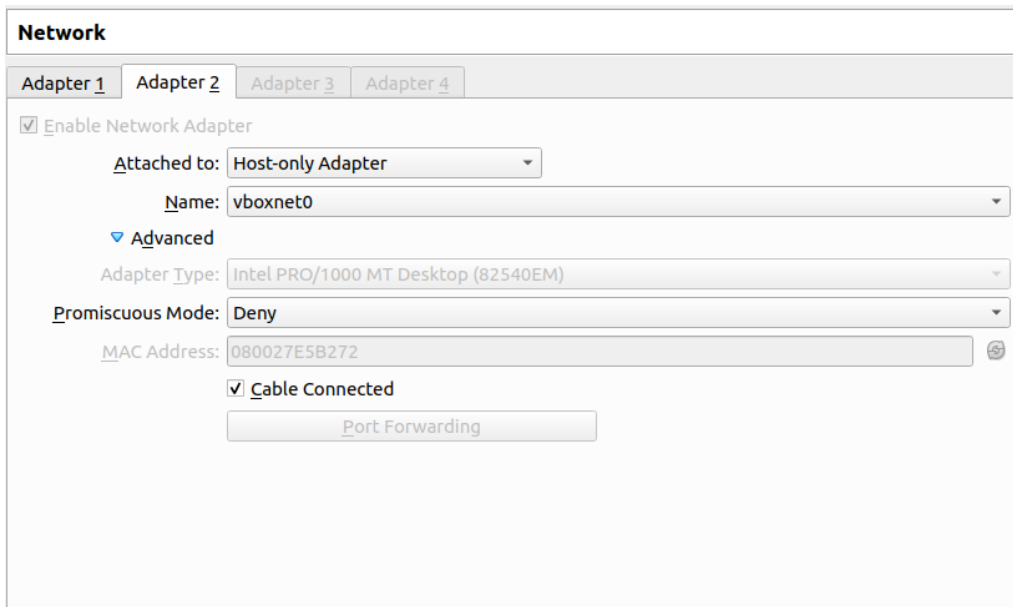


Figure 40: Virtual Machine network settings

Figure 39 gives an overview of the settings we have used when configuring the virtual machine. The machine has a base memory of 4955 MB and video memory of 64 MB, which is enough to run the brokers and gateway for our tests. For the network settings, we have used a 'Host-only' adapter, which can be seen in figure 40. As shown in the figure, there is also an adapter 1, which we have set up to allow internet access for the virtual machine. This is not strictly necessary to have in the setup.

C iPerf3

To run iPerf3, we will first need to start a server on the virtual machine. To run the server, use the following command: 'iperf3 -s'. Once the server is running on the virtual machine, open the terminal on the host machine and run the following command: 'iperf3 -c IP address', where the address of your virtual machine replaces the IP address.

```
[ 5] local 192.168.56.101 port 5201 connected to 192.168.56.1 port 48050
[ ID] Interval          Transfer          Bitrate
[ 5]  0.00-1.00      sec  0.00 Bytes      0.00 bits/sec
[ 5]  1.00-2.00      sec  19.8 KBytes     162 Kbits/sec
[ 5]  2.00-3.00      sec  28.3 KBytes     232 Kbits/sec
[ 5]  3.00-4.00      sec  31.1 KBytes     255 Kbits/sec
[ 5]  4.00-5.00      sec  28.3 KBytes     232 Kbits/sec
[ 5]  5.00-6.00      sec  28.3 KBytes     232 Kbits/sec
[ 5]  6.00-7.00      sec  28.3 KBytes     232 Kbits/sec
[ 5]  7.00-8.00      sec  28.3 KBytes     232 Kbits/sec
[ 5]  8.00-9.00      sec  31.1 KBytes     255 Kbits/sec
[ 5]  9.00-10.00     sec  28.3 KBytes     232 Kbits/sec
[ 5] 10.00-11.00     sec  28.3 KBytes     232 Kbits/sec
[ 5] 11.00-12.00     sec  31.1 KBytes     255 Kbits/sec
[ 5] 12.00-13.00     sec  28.3 KBytes     232 Kbits/sec
[ 5] 13.00-14.00     sec  29.7 KBytes     243 Kbits/sec
[ 5] 14.00-14.37     sec  11.3 KBytes     249 Kbits/sec
-----
[ ID] Interval          Transfer          Bitrate
[ 5]  0.00-14.37     sec  380 KBytes     217 Kbits/sec
-----
Server listening on 5201
-----
receiver
```

Figure 41: Results from running Iperf3 in the terminal

Figure 41 shows the results from running iPerf3. In this figure, iPerf3 was run after starting a test using the SATCOM network model. We can see that the test with iPerf3 ran for 14.36 seconds, and the bitrate was 217 kbit/s, which is slightly below the limitation set by Netem, which is 250 kbit/s. The reason for this is other traffic on the network from the test currently running. If we see that the results are far outside what we would expect, then we know that the Netem configuration has not been properly set and should restart the test.

D MQTT-SN Setup

This section describes the setup for running MQTT-SN with our analysis tool. This mainly involves setting up the gateway to receive and send messages from the clients.

The gateway we have used during this thesis is the paho mqtt-sn gateway which can be found on GitHub. In order to set up the gateway, we need to do the following steps:

1. download and unzip the files from the Github
2. Run the command 'cd paho.mqtt-sn.embedded-c/MQTTSNGateway'
3. Run the command './build.sh udp'
4. Run the command 'cd bin'
5. Finally, run the command './MQTT-SNGateway', optionally add 'gateway.conf' at the end of the command.

A terminal window with a dark purple background and white text. The output shows the MQTT-SN Gateway startup sequence. It starts with a series of asterisks, followed by the text: '* MQTT-SN Gateway', '* Part of Project Paho in Eclipse', '* (https://github.com/eclipse/paho.mqtt-sn.embedded-c.git)', and '*'. Below this, it shows '* Author : Tomoaki YAMAGUCHI' and '* Version: 1.5.1'. Another series of asterisks follows. Then, configuration details are listed: 'ConfigFile : ./gateway.conf', 'ClientList : /path/to/your_clients.conf', 'Broker : 127.0.0.1 : 1883, 8883', 'RootCApath : (null)', 'RootCAfile : (null)', 'CertKey : (null)', 'PrivateKey : (null)', 'SensorN/W : UDP Multicast 225.1.1.1:1883, Gateway Port:10000, TTL:1', and 'Max Clients : 5000'. The final line of output is '20220601 041205.014 PahoGateway-01 starts running.'

```
*****
* MQTT-SN Gateway
* Part of Project Paho in Eclipse
* (https://github.com/eclipse/paho.mqtt-sn.embedded-c.git)
*
* Author : Tomoaki YAMAGUCHI
* Version: 1.5.1
*****
ConfigFile : ./gateway.conf
ClientList : /path/to/your_clients.conf
Broker      : 127.0.0.1 : 1883, 8883
RootCApath  : (null)
RootCAfile  : (null)
CertKey     : (null)
PrivateKey   : (null)
SensorN/W   : UDP Multicast 225.1.1.1:1883, Gateway Port:10000, TTL:1
Max Clients : 5000

20220601 041205.014 PahoGateway-01 starts running.
```

Figure 42: The MQTT-SN gateway running in the terminal

The gateway should now be up and running and will look like what can be seen in figure 42. If we run the MQTT broker on a different port than the standard 1883, we will need to change this in the gateway.conf file. The value that needs to be changed is the one for 'BrokerPortNo'. We had some issues with the number of clients exceeding the maximum allowed for the gateway during testing. This mainly happened with the more challenging network models. This may have been because the clients disconnected several times during the test, and they would reconnect as new clients, adding to the current client count. For this reason, the value for 'MaxNumberOfClients' should be increased. We have used a value of 5000, but this may be excessive. No other changes should be necessary to replicate the test setup we have used.

E ZeroMQ with PGM Setup

This section describes the configurations needed to get PGM working with ZeroMQ.

ZeroMQ does not come with support for PGM through the standard installation of the `zmq` package. Enabling PGM requires a custom installation which involves a few steps. The following steps should install ZeroMQ and enable the use of PGM for the clients. We followed this process on both the local and virtual machines before testing the protocol. This process may require that we first uninstall `zmq` through a `'pip uninstall zmq'` process. Note that some of the commands may require the `'sudo'` prefix.

1. First, we need to download the ZeroMQ package from the ZeroMQ Github. We have used the newest version, `zeromq-4.1.8.tar.gz`.
2. Run the command `'tar zxvf zeromq-4.1.8.tar.gz'` in the directory where you have downloaded the package to.
3. Run the command `'cd zeromq-4.1.8'`
4. Run the command `'apt-get install libpgm-dev'`
5. Run the command `'./configure --with-pgm && make && make install'`
6. Finally, run the command `'pip install --no-binary :all: pyzmq'`

ZeroMQ should now be configured and usable with the PGM protocol. Note that PGM requires raw IP sockets that require certain privileges on Linux. We instead opted to use EPGM, which does not require any special privileges.

```
self.socket.connect("epgm://vboxnet0;224.0.0.1:6788")
```

Listing 4: Connecting to broker from client

Listing 4 show the connection string that is used to connect to the broker with PGM. It is similar to using TCP, but there are a few changes. The protocol is now set to `'epgm'` instead of `'tcp'`. The endpoint, after the `'//'` now consists of an interface (in our case `vboxnet0`), followed by a semicolon, then comes the multicast address, a colon, and the port number. Note that EPGM will not work on a loopback adapter on Linux, meaning that we can not send messages to ourselves on the same machine, requiring either another physical machine or a virtual machine. The virtual machine we have used is set up as a `'Host-only'` network, but it should also work with a `'Bridged'` network.

```
xpub_addr = 'epgm://192.168.56.101;224.0.0.1:6788'  
xsub_addr = 'epgm://224.0.0.1:6787'
```

Listing 5: Binding the endpoints on the broker

Listing 5 shows the addressing on the broker to bind the endpoints. The connection string is similar to that which is found on the client. The interface is different because it uses the local/host machine interface. ZeroMQ accepts the interface as either a name specified by the operating system or an IPv4 address in its numerical representation. In listing 4 we have used the name, while in listing 5 we have used the IPv4 representation.

F Results for MQTT

F.1 5G

Table 40 shows the results from the tests using the 5G model on the network layer. From the results we see that the test with GPS data produced 8702 packets, while the test with image data produced 8714 packets. The test with GPS data had a data rate of 24 kbit/s while the test with image data had a the data rate of 69 kbit/s. GPS data had a minimum message size of 477 bytes, an average of 482 bytes and a maximum of 484 bytes. Image data had a minimum message size of 90 bytes, an average 1553 bytes and a maximum of 5885 bytes. The test with GPS data had 0 retransmissions and 1 duplicate ACK. The test with image data produced 0 retransmissions and 0 duplicate ACKs.

Data Type	GPS	Image
Total packets	8702	8714
Data rate	24 kbit/s	69 kbit/s
Min message size	477 bytes	90 bytes
Avg message size	482 bytes	1553 bytes
Max message size	484 bytes	5885 bytes
TCP retransmissions	0	0
Duplicate ACKs	1	0

Table 40: Results from 5G tests on the network layer

Table 41 shows the results from the tests using the 5G model on the application layer. From the results using GPS data we see that 536 messages were sent with QoS 0, 532 messages with QoS 1 and 531 messages with QoS 2. Zero packet loss was recorded for all QoS levels. The minimum delay was 0.04 seconds for all QoS levels. QoS 0 and QoS 1 had an average delay of 0.07 seconds, while QoS 2 had an average delay of 0.09 seconds. No clients disconnected during the test. From the results using image data we see that 532 messages were sent using QoS 0, 533 messages with QoS 1 and 533 messages with QoS 2. Zero packet loss was recorded for all QoS levels. The minimum delay was 0.04 seconds for all QoS levels. QoS 0 had an average delay of 0.09 seconds, QoS 1 had an average delay of 0.08 seconds and QoS 2 had an average delay of 0.11 seconds. The maximum delay using QoS 0 was 1.32 seconds, 1.18 seconds using QoS 1 and 2.21 seconds using QoS 2. No clients were disconnected during the test.

Data type	QoS	Message sent	Packet loss	Min delay	Avg delay	Max delay	Disconnections
GPS	0	536	0%	0.04 s	0.07 s	0.99 s	0
	1	532	0%	0.04 s	0.07 s	1.23 s	0
	2	531	0%	0.04 s	0.09 s	0.63 s	0
	Total/Avg	1599	0%	0.04 s	0.08 s	0.95 s	0
Image	0	532	0%	0.04 s	0.09 s	1.32 s	0
	1	533	0%	0.04 s	0.08 s	1.18 s	0
	2	533	0%	0.04 s	0.11 s	2.21 s	0
	Total/Avg	1598	0%	0.04 s	0.09 s	1.57 s	0

Table 41: Results from 5G tests on the application layer

F.2 Tactical Broadband

Table 42 shows the results from testing with Tactical Broadband on the network layer. The total number of packets captured was 8816 for GPS data and 9260 for image data. The data rate produced by GPS data was 25 kbit/s, and 40 kbit/s for image data. The average message size was 477 bytes for GPS and 521 bytes for images. With GPS data

the number of retransmissions were 82 and the number of duplicate ACKs were 27. Image data produced 121 retransmissions and 61 duplicate ACKs.

Data Type	GPS	Image
Total packets	8816	9260
Data rate	25 kbit/s	40 kbit/s
Min message size	477 bytes	90 bytes
Avg message size	482 bytes	521 bytes
Max message size	484 bytes	1589 bytes
TCP retransmissions	82	121
Duplicate ACKs	27	61

Table 42: Results from Tactical Broadband tests on the network layer

Table 43 shows the results from both tests on the application layer. Both tests sent 1620 messages each and the packet loss was zero percent. For GPS data with QoS 0 and QoS 1, the average transmit delay was 0.12 seconds, while it was 0.33 seconds for QoS 2. The lowest delay found was 0.11 seconds for QoS 0 and 1, while the highest delay was 0.74 seconds using QoS 2. Using image data with QoS 0 gave an average delay of 0.15 seconds, 0.16 seconds with QoS 1 and 0.41 seconds for QoS 2. The lowest delay found was using QoS 0 and 1 with 0.11 seconds, while the highest delay was found using QoS 2 at 1.44 seconds. None of the clients disconnect during both tests.

Data type	QoS	Message sent	Packet loss	Min delay	Avg delay	Max delay	Disconnections
GPS	0	540	0%	0.11 s	0.12 s	0.45 s	0
	1	540	0%	0.11 s	0.12 s	0.44 s	0
	2	540	0%	0.31 s	0.33 s	0.74 s	0
	Total/Avg	1620	0%	0.18 s	0.19 s	0.54 s	0
Image	0	540	0%	0.11 s	0.15 s	0.74 s	0
	1	540	0%	0.11 s	0.16 s	0.58 s	0
	2	539	0%	0.32 s	0.41 s	1.44 s	0
	Total/Avg	1619	0%	0.18 s	0.24 s	0.92 s	0

Table 43: Results from Tactical Broadband tests on the application layer

The following network tests will display transmit delay data in a barchart. This was not included in tests using the 5G and Tactical Broadband network models due to the low times found, making them not suitable for visual presentation and comparison.

F.3 SATCOM

Table 44 shows the results from testing using SATCOM on the network layer. The total number of packets captured was 8833 for GPS data and 8824 for image data. The data rate from the test using GPS data is 23 kbit/s and 22 kbit/s for image data. The average message size for GPS data is 482 bytes and 467 bytes for images. The GPS test had 28 retransmissions and 27 duplicate ACKs, while the test with images had 162 retransmissions and 167 duplicate ACKs.

Data Type	GPS	Image
Total packets	8833	8824
Data rate	23 kbit/s	22 kbit/s
Min message size	477 bytes	90 bytes
Avg message size	482 bytes	467 bytes
Max message size	484 bytes	1589 bytes
TCP retransmissions	28	162
Duplicate ACKs	27	167

Table 44: Results from SATCOM tests on the network layer

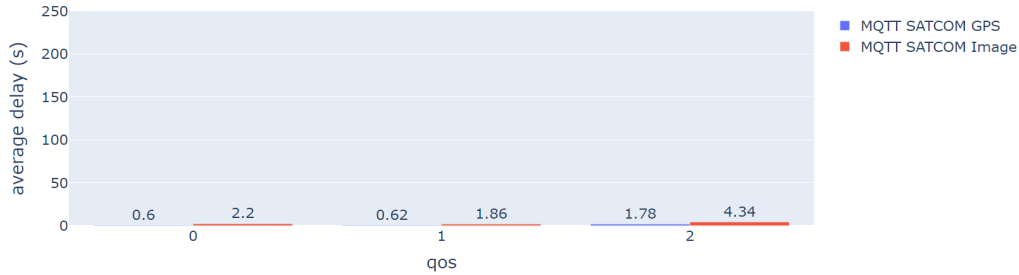


Figure 43: Average delay per QoS using the SATCOM network model

Figure 43 shows that the average delay was higher when using image data over GPS data. Table 45 shows the results from tests on the application layer with the SATCOM network model. Each test had a total of 1619 messages sent and both had a packet loss of zero percent. For the test with GPS data, the average delay was 0.60 seconds using QoS 0, 0.62 seconds using QoS 1 and 1.78 seconds using QoS 2. The lowest delay found was 0.57 seconds using QoS 0 and 1. The highest delay was found using QoS 2 and was 2.41 seconds. For the test with image data the average delay was 2.20 seconds for QoS 0, 1.86 seconds for QoS 1 and 4.34 seconds for QoS 2. The lowest delay found was 0.61 seconds using QoS 1, and the highest delay was 5.54 seconds using QoS 2. None of the clients disconnected during both of the tests.

Data type	QoS	Message sent	Packet loss	Min delay	Avg delay	Max delay	Disconnections
GPS	0	540	0%	0.57 s	0.60 s	1.16 s	0
	1	540	0%	0.57 s	0.62 s	0.74 s	0
	2	539	0%	1.69 s	1.78 s	2.41 s	0
	Total/Avg	1619	0%	0.94 s	1.00 s	1.44 s	0
Image	0	540	0%	0.63 s	2.20 s	4.35 s	0
	1	540	0%	0.61 s	1.86 s	4.38 s	0
	2	539	0%	3.16 s	4.34 s	5.54 s	0
	Total/Avg	1619	0%	1.47 s	2.80 s	4.76 s	0

Table 45: Results from SATCOM tests on the application layer

F.4 NATO Narrowband Waveform

Table 46 shows the results from testing with the NATO Narrowband model on the network layer. The total number of packets captured was 8615 for GPS data and 8541 for image data. The data rate for GPS data was 22 kbit/s and 10 kbit/s for images. The average message size was 482 bytes for GPS and 1343 bytes for images. Testing using GPS data had 313 retransmissions and 313 duplicate ACKs. For image data the number of retransmissions was 1347 and the number of duplicate ACKs was 619.

Data Type	GPS	Image
Total packets	8615	655
Data rate	22 kbit/s	4.28 kbit/s
Min message size	477 bytes	90 bytes
Avg message size	482 bytes	1078 bytes
Max message size	488 bytes	1588 bytes
TCP retransmissions	313	1424
Duplicate ACKs	313	341

Table 46: Results from NATO Narrowband tests on the network layer

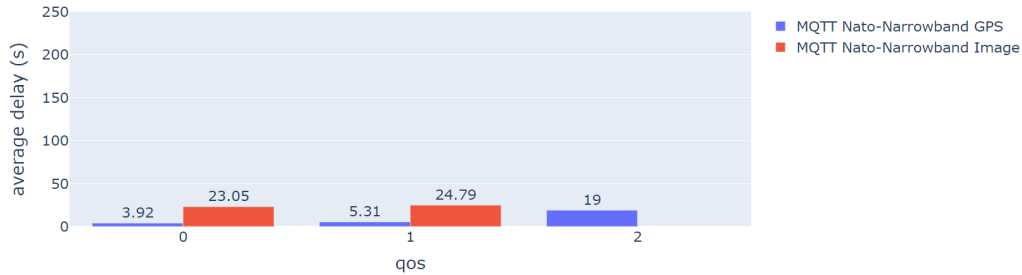


Figure 44: Average delay per QoS using the NATO Narrowband network model

Figure 44 shows that the average delay was significantly higher when using image data. It also shows that no messages arrived using QoS 2. Table 47 shows the results from the application layer. The number of messages sent using GPS data was 1605 and there was no packet loss. The average delay found was 3.92 seconds using QoS 0, 5.31 seconds using QoS 1 and 19.00 seconds using QoS 2. The smallest delay found was 1.24 seconds using QoS 0 and the highest was 40.25 seconds using QoS 2. No clients disconnected during the test with GPS data. For image data, a total of 308 messages were sent. QoS 0 had a packet loss of 85.86%, QoS 1 had a loss of 91.09% and QoS 2 lost all messages. The average delay was 23.05 seconds using QoS 0 and 24.79 seconds using QoS 1. The lowest delay found was 1.51 seconds using QoS 0 and the highest was 33.59 seconds using QoS 1. For all QoS levels, 18 clients disconnected during the test.

Data type	QoS	Message sent	Packet loss	Min delay	Avg delay	Max delay	Disconnections
GPS	0	540	(0%)	1.24 s	3.92 s	10.02 s	0
	1	533	(0%)	2.62 s	5.31 s	12.45 s	0
	2	532	(0%)	9.55 s	19.00 s	40.25 s	0
	Total/Avg	1605	(0%)	13.41 s	9.41 s	20.91 s	0
Image	0	99	85 (85.86%)	1.51 s	23.05 s	50.16 s	18
	1	101	92 (91.09%)	17.87 s	24.79 s	33.59 s	18
	2	108	108 (100%)	NA	NA	NA	18
	Total/Avg	308	285 (92.53%)	9.69 s	23.92 s	83.75 s	54

Table 47: Results from NATO Narrowband tests on the application layer

F.5 CNR with 1% loss

Table 48 shows the results on the network layer using the CNR network model with 1% loss. The total number of packets sent during testing with GPS data was 2556 and 410 with image data. GPS data had a data rate of 10 kbit/s and image data had a rate of 2.64 kbit/s. The average message size was 718 bytes using GPS data and 1041 bytes using image data. The number of retransmissions was 1957 and the number of duplicate

ACKs was 713, using GPS data. For image data the number of retransmissions was 1406 and the number of duplicate ACKs was 347.

Data Type	GPS	Image
Total packets	2556	410
Data rate	10 kbit/s	2.64 kbit/s
Min message size	81 bytes	90 bytes
Avg message size	718 bytes	1041 bytes
Max message size	2962 bytes	1588 bytes
TCP retransmissions	1957	1406
Duplicate ACKs	713	347

Table 48: Results from CNR tests with 1% loss on the network layer

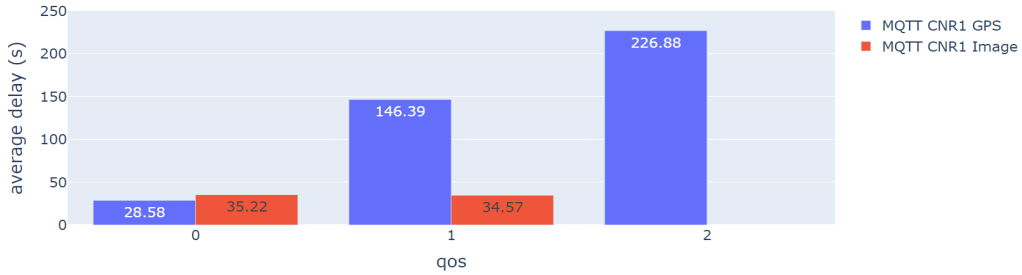


Figure 45: Average delay per QoS using the CNR network model with 1% loss

Figure 45 shows that the average delay was higher when using GPS data than image data for QoS 1. We can also see that no messages arrived for QoS 2 when sending image data. Table 49 show the results for CNR with 1% loss on the application layer. The total number of messages sent using GPS data was 829. QoS 0 had a 21.01% loss, QoS 1 had a 0% loss and QoS 2 had the highest loss with 23.26% loss. The average delay was 28.58 seconds using QoS 0, 146.39 seconds using QoS 1 and 226.88 seconds using QoS 2. The lowest delay recorded was 0.62 seconds using QoS 0 and the highest delay was 610.17 seconds using QoS 1. QoS 0 had 17 clients disconnect, while QoS 1 and 2 had 18 clients disconnect during the test. Using image data, a total of 318 messages were sent. QoS 0 had a packet loss of 91.35%, QoS 1 had a packet loss of 91.51% and QoS 2 lost all messages. The average delay for QoS 0 was 35.22 seconds and QoS 1 had an average delay of 34.57 seconds. The lowest delay found was 5.58 seconds using QoS 1 and the highest delay was 59.02 seconds using QoS 0. All QoS levels had 18 clients disconnect each during the test.

Data type	QoS	Message sent	Packet loss	Min delay	Avg delay	Max delay	Disconnections
GPS	0	357	75 (21.01%)	0.62 s	28.58 s	94.43 s	17
	1	214	0%	3.37 s	146.39 s	512.25 s	18
	2	258	60 (23.26%)	13.67 s	226.88 s	610.17 s	18
	Total/Avg	829	135 (16.28%)	5.89 s	255.90 s	227.86 s	53
Image	0	104	95 (91.35%)	6.97 s	35.22 s	59.02 s	18
	1	106	97 (91.51%)	5.58 s	34.57 s	57.64 s	18
	2	108	108 (100%)	NA	NA	NA	18
	Total/Avg	318	300 (94.34%)	6.28 s	34.90 s	58.33 s	54

Table 49: Results from CNR tests with 1% loss tests on the application layer

F.6 CNR with 10% loss

Table 50 show the test results using CNR with 1% loss on the network layer. Using GPS data the total number of packets sent was 2729 and using image data the number was 494. The data rate for tests with GPS data was 10 kbit/s and 2.84 kbit/s using image data. The average message size was 669 bytes with GPS data and 920 bytes with image data. Using GPS data the number of retransmissions was 1729 and the number of duplicate ACKs was 560. For image data the number of retransmissions was 1596 and the number of duplicate ACKs was 330.

Data Type	GPS	Image
Total packets	2729	494
Data rate	10 kbit/s	2.84 kbit/s
Min message size	76 bytes	90 bytes
Avg message size	669 bytes	920 bytes
Max message size	1717 bytes	1588 bytes
TCP retransmissions	1729	1596
Duplicate ACKs	560	330

Table 50: Results from CNR tests with 10% loss on the network layer

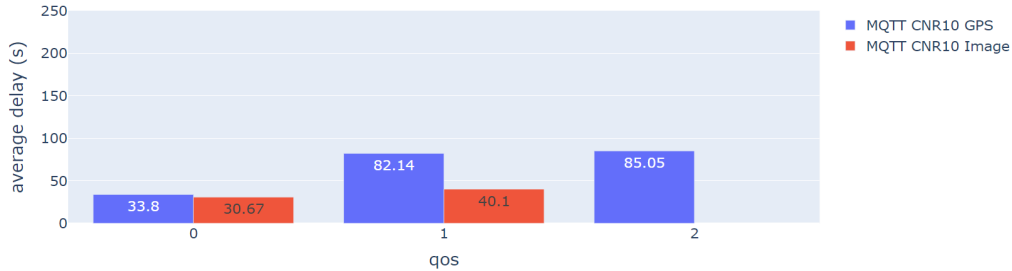


Figure 46: Average delay per QoS using the CNR network model with 10% loss

Figure 46 shows that using QoS 0, GPS data had a higher average delay over image data. Using QoS 1, the average delay was also much higher when sending GPS data. QoS 2 had no messages arrive when sending image data. Table 51 show the results from the application layer. Using GPS data the number of messages sent was 995. QoS 0 had a packet loss of 8.55%, QoS 1 had a packet loss of 19.37% and QoS 2 had a packet loss of 61.78%. The average delay for QoS 0 was 33.80 seconds, for QoS 1 it was 82.14 seconds and for QoS 2 it was 85.05 seconds. The lowest delay found was 0.55 seconds using QoS 0 and the highest delay was 500.62 seconds using QoS 1. QoS 0 had 9 clients disconnect, QoS 1 had 15 clients disconnect and QoS 2 had 18 clients disconnect. The test with image data had a total of 316 messages sent. QoS 0 had a packet loss of 94.29%, QoS 1 had a loss of 91.51% and QoS 2 lost all messages. The average delay for QoS 0 was 30.67 seconds and for QoS 1 it was 40.10 seconds. The lowest delay found was 5.30 seconds using QoS 0 and the highest delay was 58.87 seconds using QoS 1. All QoS levels ad 18 clients disconnect each.

Data type	QoS	Message sent	Packet loss	Min delay	Avg delay	Max delay	Disconnections
GPS	0	421	36 (8.55%)	0.55 s	33.80 s	120.18 s	9
	1	315	61 (19.37%)	1.19 s	82.14 s	500.62 s	15
	2	259	160 (61.78%)	4.10 s	85.05 s	185.67 s	18
	Total/Avg	995	257 (25.83%)	1.95 s	67.00 s	268.82 s	42
Image	0	105	99 (94.29%)	5.30 s	30.67 s	49.63 s	18
	1	106	97 (91.51%)	6.66 s	40.10 s	58.87 s	18
	2	105	105 (100%)	NA	NA	NA	18
	Total/Avg	316	301 (95.25%)	5.98 s	35.39 s	54.25 s	54

Table 51: Results from CNR tests with 10% loss tests on the application layer

G Results for MQTT-SN

Using MQTT-SN we have ran the same test as with MQTT. The only exceptions are that we have not tested using QoS 2 and only used 18 clients. This is due to the clients we have used do not support QoS higher than 1. There exists other clients that do support QoS 2, but they have other issues that make them unsuitable for test. Unfortunately this means that no direct comparison can be made between MQTT and MQTT-SN when it comes to QoS 2.

G.1 5G

Table 52 shows the results from the standard test using the 5G model on the network layer. The results show that 4520 packets were produced, and the data rate was 12 kbit/s. The minimum message size found was 429 bytes, the average message size was 432 bytes and the maximum was also 432 bytes.

Protocol	Standard
Total packets	4520
Data rate	12 kbit/s
Min message size	429 bytes
Avg message size	432 bytes
Max message size	432 bytes

Table 52: Results from the 5G test on the network layer (Standard)

Table 53 shows the results from the hybrid test using the 5G model on the network layer. We can see that MQTT produced 2070 packets while MQTT-SN produced 1692 packets. MQTT had a data rate of 7.35 kbit/s and MQTT-SN had a data rate of 6.69 kbit/s. With MQTT the minimum message size was 480 bytes, the average was 484 bytes and the maximum was 485 bytes. MQTT-SN had a minimum message size of 429 bytes, an average of 432 bytes and a maximum of 432 bytes.

Protocol	MQTT	MQTT-SN
Total packets	2070	1692
Data rate	7.35 kbit/s	6.69 kbit/s
Min message size	480 bytes	429 bytes
Avg message size	484 bytes	432 bytes
Max message size	485 bytes	432 bytes

Table 53: Results from the 5G test on the network layer (Hybrid)

Table 54 shows the results from using the 5G model on the application layer. The standard test produced 1076 messages with zero packet loss. The minimum delay was 0.07 seconds using both QoS levels. The average delay was 0.09 seconds for QoS 0, and 0.10 seconds for QoS 1. The max delay was 1.27 seconds using QoS 0 and 0.82 seconds using QoS 1. From the hybrid test we see that 1080 messages were produced and that zero packet loss occurred during the test. The minimum delay was 0.02 seconds for both QoS levels. The average was the same for both QoS levels at 0.03 seconds. QoS 0 had a maximum delay of 0.23 seconds, while QoS 1 had a maximum delay of 0.18 seconds.

Test Variation	QoS	Messages sent	Packet loss	Min delay	Avg delay	Max delay
Standard	0	539	0%	0.07 s	0.09 s	1.27 s
	1	537	0%	0.07 s	0.10 s	0.82 s
	Total/Avg	1076	0%	0.07 s	0.10 s	1.05 s
Hybrid	0	540	0%	0.02 s	0.03 s	0.23 s
	1	540	0%	0.02 s	0.03 s	0.12 s
	Total/Avg	1080	0%	0.02 s	0.03 s	0.18 s

Table 54: Results from 5G tests on the application layer

G.2 Tactical Broadband

Table 55 shows the results from the standard test using the Tactical Broadband model on the network layer. From the results we see that 4378 packets were produced, and the data rate was 13 kbit/s. The minimum message size found was 429 bytes, the average message size was 432 bytes and the maximum message size was also 432 bytes.

Protocol	Standard
Total packets	4378
Data rate	13 kbit/s
Min message size	429 bytes
Avg message size	432 bytes
Max message size	432 bytes

Table 55: Results from the Tactical Broadband test on the network layer (Standard)

Table 56 shows the results from the hybrid test using the Tactical Broadband model on the network layer. The MQTT protocol produced 2033 packets, while the MQTT-SN protocol produced 1672 packets. MQTT had a data rate of 7.55 kbit/s, while MQTT-SN had a data rate of 6.48 kbit/s. The minimum message size found with MQTT was 480 bytes, the average was 484 bytes and the maximum was 485 bytes. With MQTT-SN the minimum message size was 429 bytes, while the average and maximum was 432 bytes.

Protocol	MQTT	MQTT-SN
Total packets	2033	1672
Data rate	7.55 kbit/s	6.48 kbit/s
Min message size	480 bytes	429 bytes
Avg message size	484 bytes	432 bytes
Max message size	485 bytes	432 bytes

Table 56: Results from the Tactical Broadband test on the network layer (Hybrid)

Table 57 shows the results using the Tactical Broadband model on the application layer. From the results we see that the standard test produced 538 messages using QoS 0 and 536 messages using QoS 1. No packet loss occurred during the test. The minimum delay was 0.15 seconds for both QoS levels. QoS 0 had an average delay of 0.15 seconds, while QoS 1 had an average delay of 0.19 seconds. The maximum delay was 1.31 seconds using QoS 0 and 0.91 seconds using QoS 1. From the hybrid test we see that QoS 0 produced 540 messages while QoS 1 produced 538 messages. There was zero packet loss during the test. Both QoS levels had a minimum delay of 0.15 seconds. QoS 0 had an average delay of 0.16 seconds, while QoS 1 had an average delay of 0.17 seconds. The maximum delay was 0.51 seconds using QoS 0, and 0.44 seconds using QoS 1.

Test Variation	QoS	Messages sent	Packet loss	Min delay	Avg delay	Max delay
Standard	0	538	0%	0.15 s	0.17 s	1.31s
	1	536	0%	0.15 s	0.19 s	0.91 s
	Total/Avg	1074	0%	0.15 s	0.18 s	1.11 s
Hybrid	0	540	0%	0.15 s	0.16 s	0.51 s
	1	538	0%	0.15 s	0.17 s	0.44 s
	Total/Avg	1078	0%	0.15 s	0.17 s	0.48 s

Table 57: Results from Tactical Broadband tests on the application layer

G.3 SATCOM

Table 58 shows the results from the standard test using the SATCOM model on the network layer. From the results we see that 4464 packets were produced and the data rate was 13 kbit/s. The minimum message size was 429 bytes, the average was 432 bytes and the maximum was also 432 bytes.

Protocol	Standard
Total packets	4464
Data rate	13 kbit/s
Min message size	429 bytes
Avg message size	432 bytes
Max message size	432 bytes

Table 58: Results from the SATCOM test on the network layer (Standard)

Table 59 shows the results from the hybrid test using the SATCOM model on the network layer. From the results we can see that MQTT produced 2080 packets, while MQTT-SN produced 1692 packets. MQTT had a data rate of 7.05 kbit/s, while MQTT-SN had a data rate of 6.49 kbit/s. MQTT had a minimum message size of 480 bytes, an average of 484 bytes and a maximum of 485 bytes. MQTT-SN had a minimum message size of 429 bytes, an average of 432 bytes and a maximum of 432 bytes.

Data Type	MQTT	MQTT-SN
Total packets	2080	1692
Data rate	7.05 kbit/s	6.49 kbit/s
Min message size	480 bytes	429 bytes
Avg message size	484 bytes	432 bytes
Max message size	485 bytes	432 bytes

Table 59: Results from SATCOM tests on the network layer (Hybrid)

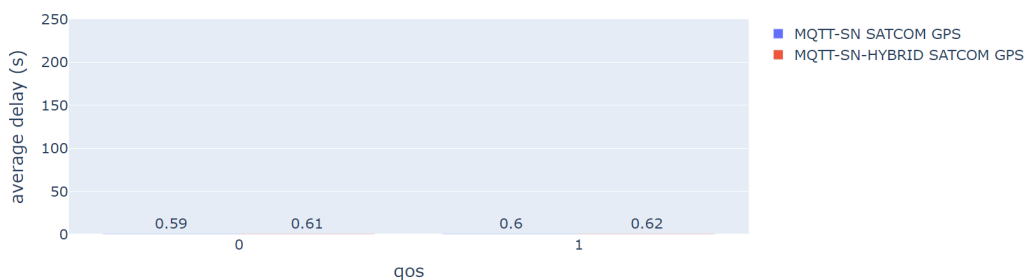


Figure 47: Average delay using the SATCOM network model

Figure 47 shows that both test had an almost identical average transmit delay. Results from the application layer is shown in table 60. From the standard test we see that QoS 0 and 1 produced 540 messages each. No loss occurred during the test. The minimum delay found was 0.57 seconds for both QoS levels. The average was the same for both QoS levels at 0.59 seconds. The max delay was 0.72 seconds for QoS 0 and 0.70 seconds for QoS 1. For the hybrid test, 540 messages were produced from both QoS levels and no packet loss was recorded. The minimum delay was 0.57 seconds for both QoS levels. The average delay was 0.61 seconds for QoS 0 and 0.62 seconds for QoS 1. The highest delay was the same at 1.02 seconds for both QoS levels.

Test Variation	QoS	Messages sent	Packet loss	Min delay	Avg delay	Max delay
Standard	0	540	0%	0.57 s	0.59 s	0.72 s
	1	540	0%	0.57 s	0.60 s	0.70 s
	Total/Avg	1080	0%	0.57 s	0.60 s	0.66 s
Hybrid	0	540	0%	0.57 s	0.61 s	1.02 s
	1	540	0%	0.57 s	0.62 s	1.02 s
	Total/Avg	1080	0%	0.57 s	0.62 s	1.02 s

Table 60: Results from SATCOM tests on the application layer

G.4 NATO Narrowband

Table 61 shows the results from the standard test using the NATO Narrowband model on the network layer. The test produced 4464 packets and had a data rate of 13 kbit/s. The minimum message size was 429 bytes, the average was 432 bytes and the maximum was also 432 bytes.

Protocol	Standard
Total packets	4464
Data rate	13 kbit/s
Min message size	429 bytes
Avg message size	432 bytes
Max message size	432 bytes

Table 61: Results from the NATO Narrowband test on the network layer (Standard)

Table 62 shows the results from the hybrid test using the NATO Narrowband model on the network layer. The MQTT protocol produced 2076 packets, while the MQTT-SN protocol produced 1692 packets. MQTT had a data rate of 6.98 kbit/s and MQTT-SN had a data rate of 6.47 kbit/s. MQTT had a minimum message size of 480 bytes, an average of 484 bytes and a maximum of 485 bytes. MQTT-SN had a minimum message size of 429 bytes, an average of 432 bytes and a maximum of 432 bytes.

Data Type	MQTT	MQTT-SN
Total packets	2076	1692
Data rate	7.20 kbit/s	6.47 kbit/s
Min message size	480 bytes	429 bytes
Avg message size	484 bytes	432 bytes
Max message size	485 bytes	432 bytes

Table 62: Results from NATO Narrowband tests on the network layer (Hybrid)

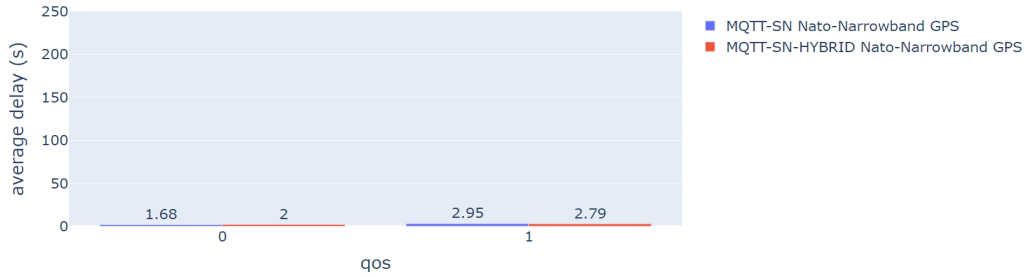


Figure 48: Average delay using the NATO Narrowband network model

Figure 48 shows that the standard test variation has a slightly higher average transmit delay than the hybrid variation. Table 63 shows the results from testing on the application layer using the NATO Narrowband network model. From the standard tests we see that both QoS levels produced 540 messages each, with no packet loss. The minimum delay for QoS 0 was 0.72 seconds, while it was 1.35 seconds for QoS 1. The average delay was 1.68 seconds for QoS 0 and 2.95 seconds for QoS 1. Max delay was found to be 3.64 seconds for QoS 0 and 4.20 seconds for QoS 1. From the hybrid tests we see that both QoS levels produced 540 messages each with no packet loss. The minimum delay was 0.72 seconds for QoS 0 and 1.04 seconds for QoS 1. The average delay for QoS 0 was 2.00 seconds, while QoS 1 had an average of 2.80 seconds. The max delay using QoS 0 was 4.13 seconds, while it was 4.92 seconds using QoS 1.

Test Variation	QoS	Messages sent	Packet loss	Min delay	Avg delay	Max delay
Standard	0	540	0%	0.72 s	1.68 s	3.64 s
	1	540	0%	1.35 s	2.95 s	4.20 s
	Total/Avg	1080	0%	1.04 s	2.32 s	3.92 s
Hybrid	0	540	0%	0.72 s	2.00 s	4.13 s
	1	540	0%	1.04 s	2.80 s	4.92 s
	Total/Avg	1080	0%	0.88 s	2.40 s	4.53 s

Table 63: Results from NATO Narrowband tests on the application layer

G.5 CNR with 1% loss

Table 64 shows the results from the standard test using the CNR model with 1% loss on the network layer. The test produced 4406 packets and had a data rate of 13 kbit/s. The minimum message size was 429 bytes, the average was 432 bytes and the maximum was also 432 bytes.

Protocol	Standard
Total packets	4406
Data rate	13 kbit/s
Min message size	429 bytes
Avg message size	432 bytes
Max message size	432 bytes

Table 64: Results from the CNR test on the network layer with 1% loss (Standard)

Table 65 shows the results from the hybrid test using the CNR model with 1% loss on the network layer. The results show that 2033 MQTT packets were produced, while MQTT-SN produced 1679 packets. MQTT had a data rate of 7.56 kbit/s, while MQTT-SN had a data rate of 6.54 kbit/s. MQTT had a minimum message size of 480 bytes, an average

of 484 bytes and a maximum of 485 bytes. MQTT-SN had a minimum message size of 429 bytes, an average of 432 bytes and a maximum of 432 bytes.

Data Type	MQTT	MQTT-SN
Total packets	2033	1679
Data rate	7.56 kbit/s	6.54 kbit/s
Min message size	480 bytes	429 bytes
Avg message size	484 bytes	432 bytes
Max message size	485 bytes	432 bytes

Table 65: Results from CNR tests on the network layer with 1% loss (Hybrid)

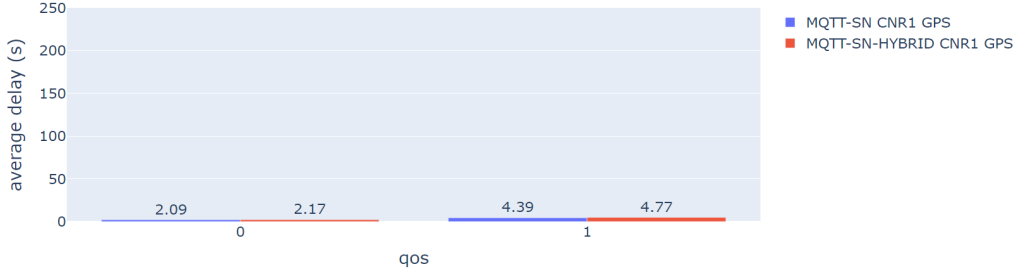


Figure 49: Average delay using the CNR network model with 1% loss

Figure 49 shows that both test had similar average transmit delay. Table 66 shows the results from tests on the application layer. From the standard test we see that QoS 0 produced 480 messages and had a packet loss of 1.11%, while QoS 1 produced 540 messages and had a packet loss of 0.74%. The minimum delay found was 0.47 seconds using QoS 0 and 1.24 seconds using QoS 1. The average delay was 2.09 seconds for QoS 0 and 4.39 seconds for QoS 1. QoS 0 had a maximum delay of 4.38 seconds. For the hybrid test we see that both QoS levels produced 450 messages each where QoS 0 had a packet loss of 0.93% and QoS 1 had a packet loss of 1.11%. The minimum delay for QoS 0 was 0.47 seconds, and 2.53 seconds for QoS 1. The average delay was 2.17 seconds for QoS 0 and 4.77 seconds for QoS 1. The maximum delay was 7.12 seconds for QoS 0 and 8.12 seconds for QoS 1.

Test Variation	QoS	Messages sent	Packet loss	Min delay	Avg delay	Max delay
Standard	0	540	6 (1.11%)	0.47 s	2.09 s	4.38 s
	1	540	4 (0.74%)	1.24 s	4.39 s	6.40 s
	Total/Avg	1080	10 (0.93%)	0.86 s	3.24 s	5.39 s
Hybrid	0	540	5 (0.93%)	0.47 s	2.17 s	7.12 s
	1	540	6 (1.11%)	2.53 s	4.77 s	8.12 s
	Total/Avg	1080	11 (1.02%)	1.50 s	3.47 s	7.62 s

Table 66: Results from CNR tests on the application layer with 1% loss

G.6 CNR with 10% loss

Table 67 shows the results from the standard test using the CNR model with 10% loss on the network layer. The test produced 1672 packets and the data rate was 4.09 kbit/s. The minimum message size was 430 bytes, the average was 432 bytes and the maximum was also 432 bytes.

Protocol	Standard
Total packets	1672
Data rate	4.09 kbit/s
Min message size	430 bytes
Avg message size	432 bytes
Max message size	432 bytes

Table 67: Results from the CNR test on the network layer with 10% loss (Standard)

Table 68 shows the results from the hybrid test using the CNR model with 10% loss on the network layer. The MQTT protocol produced 1370 packets, while the MQTT-SN protocol produced 1200. MQTT had a data rate of 4.23 kbit/s and MQTT-SN had a data rate of 2.37 kbit/s. MQTT had a minimum message size of 480 bytes, an average of 483 bytes and a maximum of 485 bytes. MQTT-SN had a minimum message size of 429 bytes, an average of 432 bytes and a maximum of 432 bytes.

Data Type	MQTT	MQTT-SN
Total packets	1370	1200
Data rate	4.23 kbit/s	4.42 kbit/s
Min message size	480 bytes	429 bytes
Avg message size	483 bytes	432 bytes
Max message size	485 bytes	432 bytes

Table 68: Results from CNR tests on the network layer with 10% loss (Hybrid)

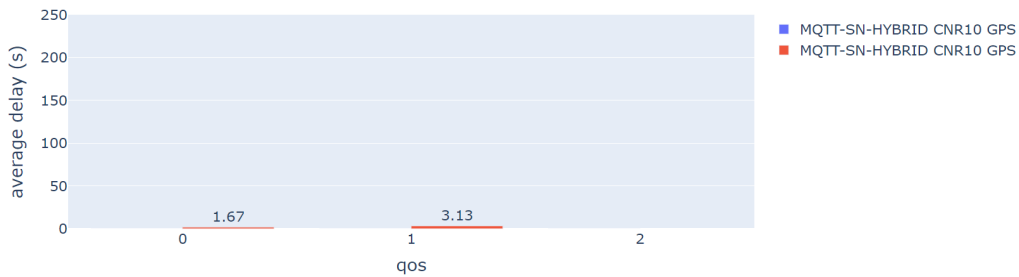


Figure 50: Average delay using the CNR network model with 10% loss

Figure 50 shows that no messages were received while running the standard test. Table 69 shows the results from tests using the CNR model with 10% loss on the application layer. From the standard test we can see that 840 messages were produced. During the test all of the messages were lost. From the hybrid test, 480 messages were sent using QoS 0 and 360 messages were sent using QoS 1. 20.21% of the messages were lost using QoS 0 and 25% of the messages were lost using QoS 1. The minimum delay was 0.46 seconds using QoS 0 and 1.10 seconds using QoS 1. The average delay was 1.67 seconds using QoS 0 and 3.13 seconds using QoS 1. The maximum delay was 5.78 seconds using QoS 0 and 5.00 seconds using QoS 1.

Test Variation	QoS	Messages sent	Packet loss	Min delay	Avg delay	Max delay
Standard	0	420	420 (100%)	NA	NA	NA
	1	420	420 (100%)	NA	NA	NA
	Total/Avg	840	840 (100%)	NA	NA	NA
Hybrid	0	480	97 (20.21%)	0.46 s	1.67 s	5.78 s
	1	360	90 (25.00%)	1.10 s	3.13 s	5.00 s
	Total/Avg	840	187 (22.26%)	0.78 s	2.40 s	5.39 s

Table 69: Results from CNR tests on the application layer with 10% loss

H Results for ZeroMQ with TCP

H.1 5G

Table 70 shows the results from testing using the 5G model on the network layer. From the results we can see that the test using GPS data produced 3486 packets, while the test using image data produced 3445 packets. The test with GPS had a data rate of 20 kbit/s and image data had a data rate of 19 kbit/s. The test using GPS data had a minimum message size of 485 bytes, an average of 488 bytes and a maximum of 490 bytes. The test using image data had a minimum message size of 98 bytes, an average of 497 bytes and a maximum of 1595 bytes. Zero retransmissions and duplicate ACKs were found in both tests.

Data Type	GPS	Image
Total packets	3486	3445
Data rate	20 kbit/s	19 kbit/s
Min message size	485 bytes	98 bytes
Avg message size	488 bytes	497 bytes
Max message size	490 bytes	1595 bytes
TCP retransmissions	0	0
Duplicate ACKs	0	0

Table 70: Results from 5G tests on the network layer

Table 71 shows the results from testing using the 5G model on the application layer. The test using GPS data produced 1593 messages and had zero packet loss. The minimum delay was 0.03 seconds, the average was 0.12 seconds and the maximum delay was 0.39 seconds. The test using image data produced 1566 messages with zero packet loss. The minimum delay was 0.05 seconds, the average delay was 0.36 seconds and the maximum delay was 0.75 seconds.

Data type	Messages sent	Packet loss	Min delay	Avg delay	Max delay
GPS	1593	0%	0.03 s	0.12 s	0.39 s
Image	1566	0%	0.05 s	0.36 s	0.75 s

Table 71: Results from 5G tests on the application layer

H.2 Tactical Broadband

Table 72 shows the results from testing the Tactical Broadband model on the network layer. Using GPS data, 3534 packets were produced, while image data produced 3017 packets. The data rate using GPS data was 20 kbit/s and 18 kbit/s for image data. GPS had a minimum message size of 485 bytes, an average of 488 bytes and a maximum of 490 bytes. The test using image data had a minimum message size of 98 bytes, an average size of 500 bytes and a maximum of 1595 bytes. The test with GPS data produced 24 retransmissions and 0 duplicate ACKs, while image data produced 73 retransmissions and 38 duplicate ACKs.

Data Type	GPS	Image
Total packets	3534	3105
Data rate	20 kbit/s	18 kbit/s
Min message size	485 bytes	98 bytes
Avg message size	488 bytes	500 bytes
Max message size	490 bytes	1595 bytes
TCP retransmissions	24	73
Duplicate ACKs	0	38

Table 72: Results from Tactical Broadband tests on the network layer

Table 73 shows the results on the application layer. The test using GPS data produced 1593 messages where zero messages were lost. The minimum delay was 0.11 seconds, the average delay was 0.21 seconds and the maximum delay was 0.73 seconds. The test using image data produced 1566 messages where zero messages were lost. The minimum delay was 0.14 seconds, the average delay was 0.47 seconds and the maximum delay was 1.41 seconds.

Data type	Messages sent	Packet loss	Min delay	Avg delay	Max delay
GPS	1593	0%	0.11 s	0.21 s	0.73 s
Image	1566	0%	0.14 s	0.47 s	1.41 s

Table 73: Results from Tactical Broadband tests on the application layer

The following network tests will display transmit delay data in a barchart. This was not included in tests using the 5G and Tactical Broadband network models due to the low times found, making them not suitable for visual presentation and comparison.

H.3 SATCOM

Table 74 show the results from using the SATCOM model on the network layer. Using GPS data 3448 packets were produced, while image data produced 3372 packets. The data rate when testing with GPS data was 19 kbit/s and 20 kbit/s for image data. GPS had a minimum message size of 485 bytes, an average size of 488 bytes and a maximum size of 490 bytes. Image data had a minimum message size of 98 bytes, an average of 506 bytes and a maximum of 1595 bytes. The test using GPS data produced 0 retransmissions and 0 duplicate ACK, while image data produced 71 retransmissions and 71 duplicate ACKs.

Data Type	GPS	Image
Total packets	3448	3372
Data rate	19 kbit/s	20 kbit/s
Min message size	485 bytes	98 bytes
Avg message size	488 bytes	506 bytes
Max message size	490 bytes	1595 bytes
TCP retransmissions	0	71
Duplicate ACKs	0	71

Table 74: Results from SATCOM tests on the network layer

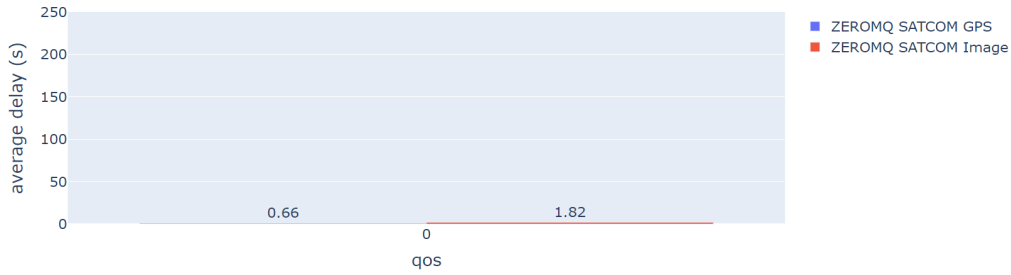


Figure 51: Average delay using the SATCOM network model

Figure 51 shows that image data had a much higher average transmit delay than GPS data. Table 75 shows the results from testing on the application layer. The test using GPS data produced 1593 messages and had a packet loss of zero percent. The minimum delay was 0.58 seconds, the average delay was 0.66 seconds and the maximum delay was 1.06 seconds. The test using image data produced 1566 messages where zero messages were lost. The minimum delay was 0.63 seconds, the average was 1.82 seconds and the maximum was 3.59 seconds.

Data type	Messages sent	Packet loss	Min delay	Avg delay	Max delay
GPS	1593	0%	0.58 s	0.66 s	1.06 s
Image	1566	0%	0.63 s	1.82 s	3.59 s

Table 75: Results from SATCOM tests on the application layer

H.4 NATO Narrowband

Table 76 shows the results from testing using the NATO Narrowband model on the network layer. GPS data produced 1645 packets and image data produced 876 packets. The data rate from using GPS was 9.80 kbit/s and 3.69 kbit/s from using image data. GPS had a minimum message size of 485 bytes, an average of 488 bytes and a maximum of 490 bytes. Image data had a minimum message size of 98 bytes, an average of 1044 bytes and a maximum of 1514 bytes. The test using GPS data produced 95 retransmissions and 137 duplicate ACKs, while image data produced 242 retransmissions and 248 duplicate ACKs.

Data Type	GPS	Image
Total packets	1645	500
Data rate	9.80 kbit/s	3.69 kbit/s
Min message size	485 bytes	98 bytes
Avg message size	488 bytes	1044 bytes
Max message size	490 bytes	1514 bytes
TCP retransmissions	95	242
Duplicate ACKs	137	248

Table 76: Results from NATO Narrowband tests on the network layer



Figure 52: Average delay using the NATO Narrowband network model

Figure 52 shows that image data had a significantly higher average transmit delay than GPS data. Table 77 shows the results from testing on the application layer. The test using GPS data produced 1593 messages, where zero message were lost. The minimum delay was 0.76 seconds, the average was 2.83 seconds and the maximum delay was 5.81 seconds. The test using image data produced 1566 messages, where 1199 messages were lost. The minimum delay was 1.54 seconds, the average delay was 269.58 seconds and the maximum delay was 515.58 seconds.

Data type	Messages sent	Packet loss	Min delay	Avg delay	Max delay
GPS	1593	0%	0.76 s	2.83 s	5.81 s
Image	1566	1199 (76.56%)	1.54 s	269.58 s	515.58 s

Table 77: Results from NATO Narrowband tests on the application layer

H.5 CNR with 1% loss

Table 78 shows the results from testing using the CNR model with 1% loss on the network layer. The test using GPS data produced 1508 packets, while image data produced 1048 packets. The data rate using GPS was 8.51 kbit/s, and 12 kbit/s for image data. The test using GPS data had a minimum message size of 306 bytes, an average of 493 bytes and a maximum of 1514 bytes. The test using image data had a minimum message size of 82 bytes, an average of 1359 bytes and a maximum of 1514 bytes. GPS data produced 154 retransmissions and 256 duplicate ACKs, while image data produced 272 retransmissions and 340 duplicate ACKs.

Data Type	GPS	Image
Total packets	1508	1048
Data rate	8.51 kbit/s	12 kbit/s
Min message size	306 bytes	82 bytes
Avg message size	493 bytes	1359 bytes
Max message size	1514 bytes	1514 bytes
TCP retransmissions	154	272
Duplicate ACKs	256	340

Table 78: Results from CNR tests with 1% loss on the network layer

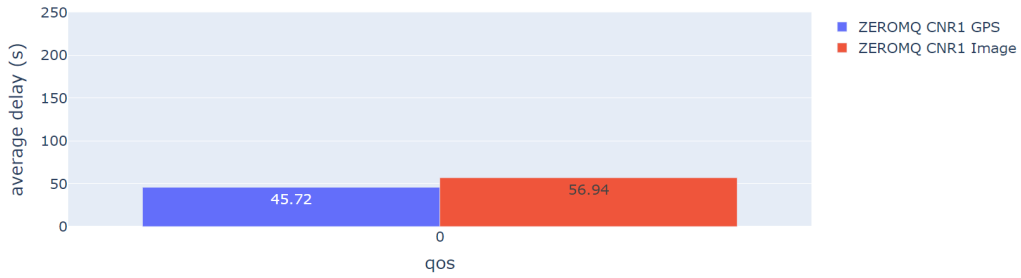


Figure 53: Average delay using the CNR network model with 1% loss

Figure 53 shows that image data had a significantly higher average transmit delay than GPS data. Table 79 shows the results from testing on the application layer. The test using GPS data produced 1593 messages, where the packet loss was 8.35%. The minimum delay was 0.91 seconds, the average delay was 45.72 seconds and the maximum delay was 162.75 seconds. The test using image data produced 1566 messages, and the packet loss was 95.02%. The minimum delay was 6.88 seconds, the average delay was 56.94 seconds and the maximum delay was 139.07 seconds.

Data type	Messages sent	Packet loss	Min delay	Avg delay	Max delay
GPS	1593	133 (8.35%)	0.91 s	45.72 s	162.75 s
Image	1566	1488 (95.02%)	6.88 s	56.94 s	139.07 s

Table 79: Results from CNR tests on the application layer with 1% loss

H.6 CNR with 10% loss

Table 80 shows the results from testing using the CNR model with 10% loss on the network layer. The test using GPS data produced 1597 packets, while the test using image data produced 563 packets. The data rate from the GPS test was 9.58 kbit/s, and 3.94 kbit/s for image data. GPS data had a minimum message size of 304 bytes, an average of 532 bytes and a maximum of 1754 bytes. The test using image data had a minimum message size of 98 bytes, an average of 1166 bytes and a maximum of 1595 bytes. The GPS test produced 232 retransmissions and 471 duplicate ACKs, while the Image test produced 335 retransmissions and 286 duplicate ACKs.

Data Type	GPS	Image
Total packets	1597	563
Data rate	9.58 kbit/s	3.94 kbit/s
Min message size	304 bytes	98 bytes
Avg message size	532 bytes	1166 bytes
Max message size	1754 bytes	1595 bytes
TCP retransmissions	232	335
Duplicate ACKs	471	286

Table 80: Results from CNR tests with 10% loss on the network layer

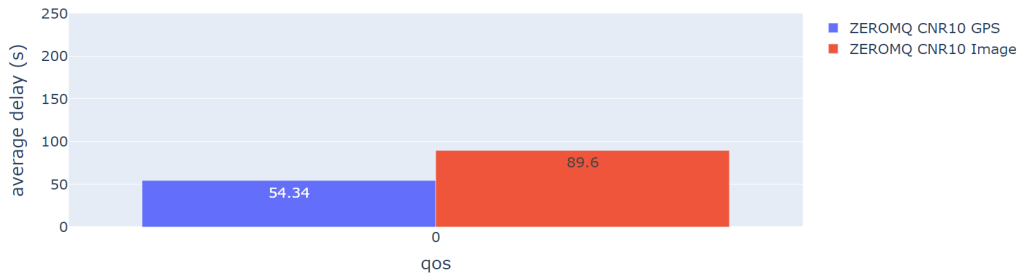


Figure 54: Average delay using the CNR network model with 10% loss

Figure 54 shows that the average transmit delay was much higher for image data over GPS data. Table 81 shows the results from testing on the application layer. The test using GPS data produced 1593 messages and the packet loss was 8.35%. The minimum delay was 5.73 seconds, the average delay was 54.34 seconds and the maximum delay was 185.58 seconds. The test using image data produced 1556 messages and the packet loss was 97.30%. The minimum delay was 8.97 seconds, the average delay was 89.60 seconds and the maximum delay was 398.51 seconds.

Data type	Messages sent	Packet loss	Min delay	Avg delay	Max delay
GPS	1593	133 (8.35%)	5.73 s	54.34 s	185.58 s
Image	1556	1514 (97.30%)	8.97 s	89.60 s	398.51 s

Table 81: Results from CNR tests on the application layer with 10% loss

I Results for ZeroMQ with PGM

I.1 5G

Table 82 shows the results from testing using the 5G model on the network layer. From the results we can see that the test using GPS data produced 18 222 packets, while the test using image data produced 110 357 packets. the test with GPS had a data rate of 28 kbit/s and image data had a data rate of 637 kbit/s. The test using GPS data had a minimum message size of 488 bytes, an average of 491 bytes and a maximum of 493 bytes. The test using image data had a minimum message size of 171 bytes, an average of 1310 bytes and a maximum of 1494 bytes.

Data Type	GPS	Image
Total packets	18 222	110 383
Data rate	28 kbit/s	637 kbit/s
Min message size	488 bytes	171 bytes
Avg message size	491 bytes	1310 bytes
Max message size	493 bytes	1494 bytes

Table 82: Results from 5G tests on the network layer

Table 83 shows the results from testing using the 5G model on the application layer. The test using GPS data produced 1619 messages and had zero packet loss. The minimum delay was 0.03 seconds, the average was 0.10 seconds and the maximum delay was 1.03 seconds. The test using image data produced 1593 messages with zero packet loss. The minimum delay was 0.04 seconds, the average delay was 3.07 seconds and the maximum delay was 8.51 seconds.

Data type	Messages sent	Packet loss	Min delay	Avg delay	Max delay
GPS	1619	0%	0.03 s	0.10 s	1.03 s
Image	1593	0%	0.04 s	3.07 s	8.51 s

Table 83: Results from 5G tests on the application layer

I.2 Tactical Broadband

Table 84 shows the results from testing the Tactical Broadband model on the network layer. Using GPS data, 22 262 packets were produced, while image data produced 81 980 packets. The data rate using both GPS was 42 kbit/s and 502 kbit/s for image data. GPS had an minimum message size of 488 bytes, an average of 491 bytes and a maximum of 493 bytes. The test using image data had a minimum message size of 156 bytes, an average size of 1184 bytes and a maximum of 1494 bytes.

Data Type	GPS	Image
Total packets	22 262	81 980
Data rate	42 kbit/s	502 kbit/s
Min message size	488 bytes	156 bytes
Avg message size	491 bytes	1184 bytes
Max message size	493 bytes	1494 bytes

Table 84: Results from Tactical Broadband tests on the network layer

Table 85 shows the results on the application layer. The test using GPS data produced 1568 messages where zero messages were lost. The minimum delay was 0.15 seconds, the

average delay was 0.31 seconds and the maximum delay was 1.94 seconds. The test using image data produced 1543 messages where 2 messages were lost. The minimum delay was 0.18 seconds, the average delay was 1.67 seconds and the maximum delay was 6.56 seconds.

Data type	Messages sent	Packet loss	Min delay	Avg delay	Max delay
GPS	1568	0%	0.15 s	0.31 s	1.94 s
Image	1543	2 (0.13%)	0.18 s	1.67 s	6.56 s

Table 85: Results from Tactical Broadband tests on the application layer

The following network tests will display transmit delay data in a barchart. This was not included in tests using the 5G and Tactical Broadband network models due to the low times found, making them not suitable for visual presentation and comparison.

I.3 SATCOM

Table 86 show the results from using the SATCOM model on the network layer. Using GPS data 17 976 packets were produced, while image data produced 3070 packets. The data rate when testing with GPS data was 40 kbit/s and 263 kbit/s for image data. GPS had a minimum message size of 488 bytes, an average size of 491 bytes and a maximum size of 493 bytes. Image data had a minimum message size of 171 bytes, an average of 1136 bytes and a maximum of 1494 bytes.

Data Type	GPS	Image
Total packets	17 976	103 608
Data rate	40 kbit/s	263 kbit/s
Min message size	488 bytes	171 bytes
Avg message size	491 bytes	1136 bytes
Max message size	493 bytes	1494 bytes

Table 86: Results from SATCOM tests on the network layer

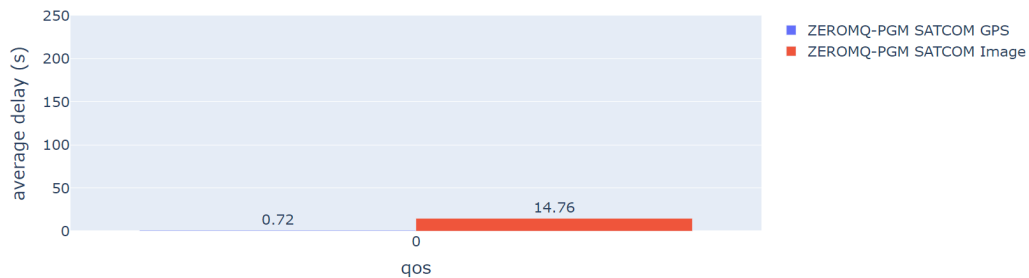


Figure 55: Average delay using the SATCOM network model

Figure 55 shows that image data had a much higher average transmit delay than GPS data. Table 87 shows the results from testing on the application layer. The test using GPS data produced 1584 messages where zero messages were lost. The minimum delay was 0.58 seconds, the average delay was 0.72 seconds and the maximum delay was 1.10 seconds. The test using image data produced 1566 messages where 12.77% of the messages were lost. The minimum delay was 0.84 seconds, the average was 18.93 seconds and the maximum was 56.52 seconds.

Data type	Messages sent	Packet loss	Min delay	Avg delay	Max delay
GPS	1584	0%	0.58 s	0.72 s	1.10 s
Image	1566	200 (12.77%)	0.84 s	14.76 s	56.52 s

Table 87: Results from SATCOM tests on the application layer

I.4 NATO Narrowband

Table 88 shows the results from testing using the NATO Narrowband model on the network layer. GPS data produced 38 878 packets and image data produced 57 618 packets. The data rate from using GPS was 45 kbit/s and 72 kbit/s from using image data. GPS had a minimum message size of 331 bytes, an average of 523 bytes and a maximum of 1494 bytes. Image data had a minimum message size of 171 bytes, an average of 186 bytes and a maximum of 1494 bytes.

Data Type	GPS	Image
Total packets	38 878	57 618
Data rate	45 kbit/s	72 kbit/s
Min message size	331 bytes	171 bytes
Avg message size	523 bytes	186 bytes
Max message size	1494 bytes	1494 bytes

Table 88: Results from NATO Narrowband tests on the network layer

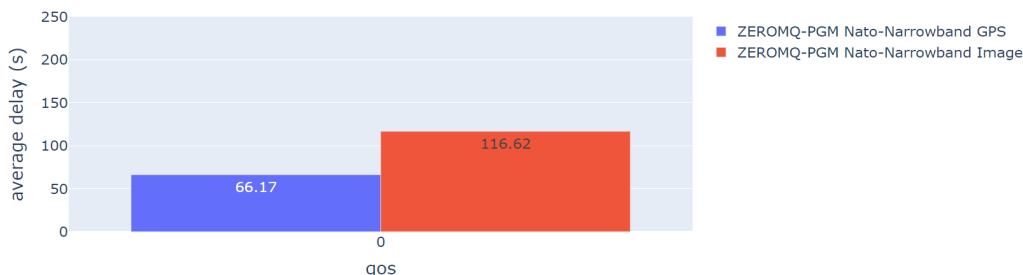


Figure 56: Average delay using the NATO Narrowband network model

Figure 56 shows that image data had a significantly higher average transmit delay than GPS data. Table 89 shows the results from testing on the application layer. The test using GPS data produced 1592 messages, where 770 messages were lost. The minimum delay was 0.89 seconds, the average was 66.17 seconds and the maximum delay was 98.18 seconds. The test using image data produced 1540 messages, where 1378 messages were lost. The minimum delay was 1.70 seconds, the average delay was 116.62 seconds and the maximum delay was 222.08 seconds.

Data type	Messages sent	Packet loss	Min delay	Avg delay	Max delay
GPS	1592	770 (48.37%)	0.89 s	66.17 s	98.18 s
Image	1540	1378 (89.48%)	1.70 s	116.62 s	222.08 s

Table 89: Results from NATO Narrowband tests on the application layer

I.5 CNR with 1% loss

Table 90 shows the results from testing using the CNR model with 1% loss on the network layer. The test using GPS data produced 49 188 packets, while image data produced 23 107 packets. The data rate using GPS was 49 kbit/s, and 27 kbit/s for image data. The test using GPS data had an minimum message size of 488 bytes, an average of 492 bytes and a maximum of 914 bytes. The test using image data had a minimum message size of 171 bytes, an average of 208 bytes and a maximum of 493 bytes.

Data Type	GPS	Image
Total packets	49 188	23 107
Data rate	49 kbit/s	27 kbit/s
Min message size	488 bytes	171 bytes
Avg message size	492 bytes	208 bytes
Max message size	914 bytes	493 bytes

Table 90: Results from CNR tests with 1% loss on the network layer

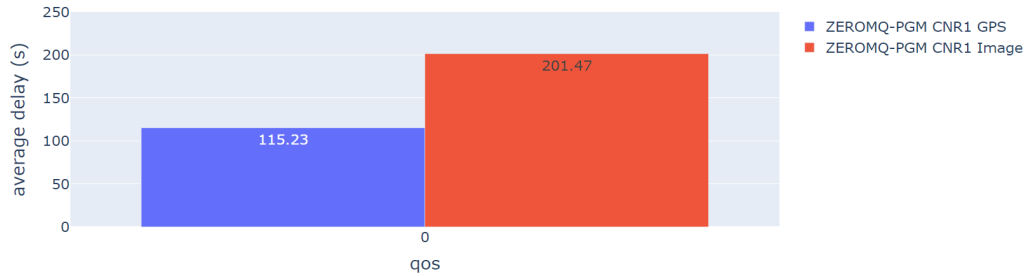


Figure 57: Average delay using the CNR network model with 1% loss

Figure 57 shows that image data had a significantly higher average transmit delay than GPS data. Table 91 shows the results from testing on the application layer. The test using GPS data produced 1566 messages, where 1157 messages were lost. The minimum delay was 1.75 seconds, the average delay was 115.23 seconds and the maximum delay was 224.01 seconds. The test using image data produced 1522 messages, where 1397 messages were lost. The minimum delay was 14.93 seconds, the average delay was 201.47 seconds and the maximum delay was 353.92 seconds.

Data type	Messages sent	Packet loss	Min delay	Avg delay	Max delay
GPS	1566	1157 (73.88%)	1.75 s	115.23 s	224.01 s
Image	1522	1397 (91.79%)	14.93 s	201.47 s	353.92 s

Table 91: Results from CNR tests on the application layer with 1% loss

I.6 CNR with 10% loss

Table 92 shows the results from testing using the CNR model with 10% loss on the network layer. The test using GPS data produced 46 543 packets, while the test using image data produced 40 872 packets. The data rate from the GPS test was 52 kbit/s, and 31 kbit/s for image data. GPS data had a minimum message size of 488 bytes, an average of 493 bytes and a maximum of 914 bytes. The test using image data had a minimum message size of 171 bytes, an average of 183 bytes and a maximum of 491 bytes.

Data Type	GPS	Image
Total packets	46 543	40 872
Data rate	52 kbit/s	31 kbit/s
Min message size	488 bytes	171 bytes
Avg message size	493 bytes	183 bytes
Max message size	914 bytes	491 bytes

Table 92: Results from CNR tests with 10% loss on the network layer

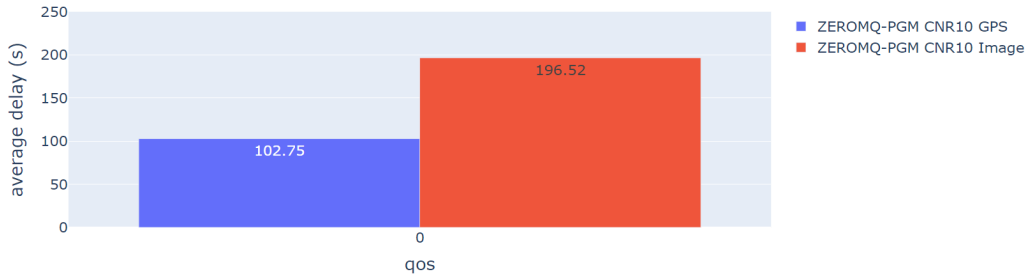


Figure 58: Average delay using the CNR network model with 10% loss

Figure 58 shows that the average transmit delay was much higher for image data over GPS data. Table 93 shows the results from testing on the application layer. The test using GPS data produced 1565 messages, where 954 messages were lost. The minimum delay was 1.30 seconds, the average delay was 102.75 seconds and the maximum delay was 140.94 seconds. The test using image data produced 1532 messages, where 1429 messages were lost. The minimum delay was 9.98 seconds, the average delay was 196.52 seconds and the maximum delay was 348.02 seconds.

Data type	Messages sent	Packet loss	Min delay	Avg delay	Max delay
GPS	1565	954 (60.96%)	1.30 s	102.75 s	140.94 s
Image	1532	1429 (93.28%)	9.98 s	196.52 s	348.02 s

Table 93: Results from CNR tests on the application layer with 10% loss

