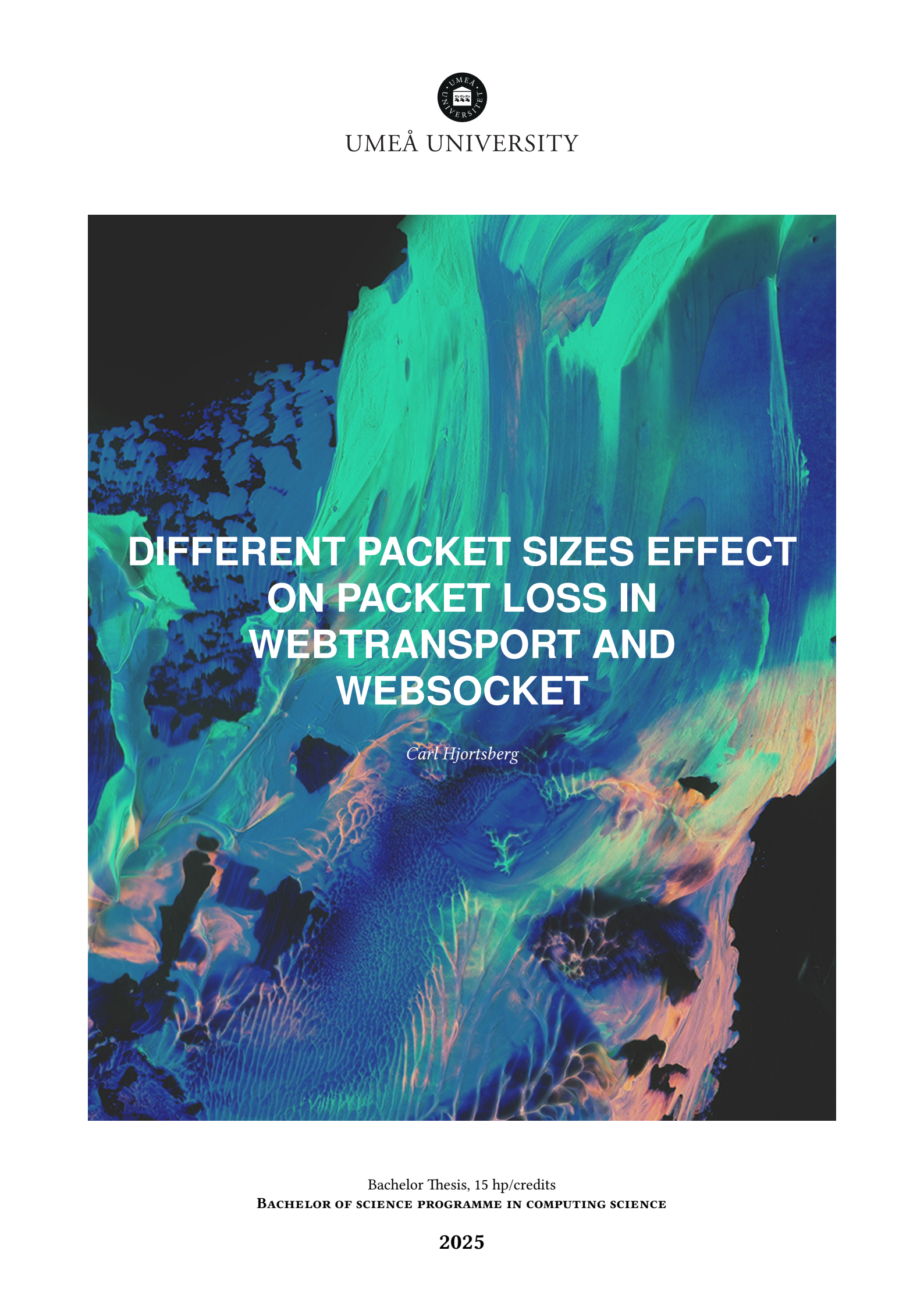

Bachelor Thesis, 15 hp/credits

Bachelor of science programme in computing science

2025

Abstract

Packages of information, or packets, are used across the internet and in computer networking to communicate between a sender and a receiver. The sizes of packets can vary depending on how much data there is to be sent. When a packet is sent from sender to receiver it can for numerous reasons be dropped, which is reffered to as packet loss. Previous work in this field have investigated if packet size has an effect on packet loss using different protocols, but there is a gap in investigating if packet size has an effect on packet loss in the protocols WebSocket and WebTransport. This thesis aims to fill that gap. This is done with the help of experiments for both WebSocket and WebTransport, where the round trip time for a message sent from a client to a server is measured. Then by increasing the size of the message, thus increasing the size of the packet, it is evaluated if the change in packet size has an effect on the packet loss. The experiments test different levels of packet loss, 10%, 15% and 20% respectively and send messages in the lengths of 1 to 1000 characters long. The produced results suggest that packet size does not affect packet loss in WebSocket and WebTransport most likely due to the fixed packet loss and test environment not benefiting either small or larger packets. Considering these findings, developers using Socket.IO with WebSocket and WebTransport can take the findings of this thesis into account to minimize packet loss when dealing with different packet sizes.

Acknowledgements

I would like to extend my deepest gratitude to my supervisor Jerry Eriksson, who has guided me through this thesis with his knowledge and support. I would also like to thank my family, friends and my girlfriend, who have supported me through the course of this thesis.

Contents

1 Introduction

To communicate in computer networking from a sender to a receiver, packages of information are used, these packages are referred to as packets [6, p.32]. The packets can have different sizes depending on how much data they contain. These packets that contain data can for several reasons be dropped, and thus not arrive at their destination as planned. For example, the output buffer at a router the packet will use could be full, resulting in packet loss, which is when a packet is dropped [6, p.52-53].

Packet loss is handled differently by different protocols. The two most common protocols, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) handle this differently. TCP is seen as a reliable protocol that makes sure that packets are delivered and that they are so in order. UDP is the opposite and it does not guarantee that packets arrive at their destination at all [6, p.220]. WebSocket and WebTransport are two other protocols that can be used when establishing a connection between a client and a server. The library SocketIO provides a client to server communication with WebSocket and WebTransport.[1]

Having too much packet loss in a connection can severely impact performance and slow down the connection. A recent bachelor thesis [4] have done an analysis on the performance of WebSocket and WebTransport in different levels of packet loss. The results showed that WebTransport was advantageous over WebSocket both in Head-of-line blocking mitigation and in performance measuring round trip time [4]. In the suggested work section of [4], the idea to consider if the performance can be impacted by packet size is proposed. Evaluating the effects of packet sizes on performance has also been done in the past. In [1] it was found that larger packet sizes improved performance of TCP-Reno connections that were dominated by link errors, and smaller packet sizes improved performance when the connection was dominated by congestion.

With these findings and the suggested work by [4] in mind, it becomes interesting to investigate how packet sizes would affect packet loss in WebSocket and WebTransport, which is what this thesis investigates. The research question posed is:

- How does different packet sizes affect packet loss in WebSocket and WebTransport?

To answer this question, several experiments were carried out inspired by previous work. In the experiments, messages of different sizes are sent in a network connection with packet loss and measuring the round trip time of those messages to see if packet size affects the packet loss. The results of these experiments were analyzed and evaluated to get an understanding of what the results mean and to help use them to answer the research question of this thesis. The results produced suggest that packet size does not have an effect on packet loss in WebSocket and WebTransport, most likely because the test environment did not benefit either larger or smaller packet sizes.

The rest of this thesis is structured as follows: Chapter 2 presents previous work in the field. Chapter 3 provides background information on the topic. Chapter 4 describes the methodology used in this thesis. Chapter 5 presents the results found. Chapter 6 discusses the results and Chapter 7 concludes the findings of the thesis.

[1] https://socket.io/docs/v4/

2 Previous Work

Previous work in this field have investigated the impact of packet size in networks with packet loss with the help of different protocols.

In [1] the effect of packet size on the performance of a single TCP-Reno connection over a lossy and congested link was evaluated. Two types of packet losses were analyzed: packet loss due to buffer overflow, and due to channel loss. To do this a system model was presented where a persistent data source was assumed. To explain the extent of packet loss in the model, the channel would switch between a ”good” and a ”bad” state. Simulations were also performed to validate the system model using a ns-2 Network Simulator. The results showed that larger packet sizes are beneficial for TCP-Reno performance when the connection is dominated by link errors. When the connection is dominated by congestion, smaller packet sizes have higher throughput [1].

The study in [5] have investigated how packet size affects loss rate and delay characteristics in a wireless real-time application. To do this a theoretical analysis was made as well as a practical experiment. In the theoretical analysis assumptions and equations were used to present an analytical model that was made for the dependency between delay characteristics and packet length. To test and verify the theoretical analysis, it was tested using experiments in an ad hoc network together with WLAN technologies. The experiments had a sender and receiver running on different machines and an application had been implemented for the experiment. In the experiments a constant flow of packets with dummy payload data was sent. The results found that by carefully designing packetization schemes in the application layer, the utilization of radio link resources in delay sensitive media streaming could be improved when running on bad wireless connections [5].

In [3] packet size distribution was investigated, more specifically, how the packet loss process is affected by packet size distribution as well as the related performance of forward error correction. To investigate this, a mathematical model was presented for the loss process of a specific queue. The model had a single queue and packets coming in to the system came from two different sources and were stored in a buffer. The results of the model was then compared to the results of simulations using several other packet size distributions. It was concluded that the distribution of packet size does affect the process of packet loss and also the efficiency of forward error correction. The results also found that the performance of forward error correction in real networks can be evaluated using analytical models of the packet size distribution that matches the first three moments of the empirical packet size distribution [3].

The study in [2] evaluated the effects of packet loss when sending MPEG compressed videos over the public internet. The protocols that were used were RTP and UDP. To do this, three sites located around the United States and Europe were set up and connected to the public internet. Each site then streamed data to a destination where the error statistics were collected. The data that was sent was two MPEG coded video clips that were 5 minutes long. Measurements were taken over several weeks, at any hour between the test sites. The results showed several different error characteristics, such as out of order packet reception, probabilities of conditional packet loss and packet loss affected by packet size [2].

The bachelor thesis [4] evaluated the performance of the network protocols WebSocket

and WebTransport using Socket.IO and this was done under varying network conditions. The evaluations consisted of two experiments conducted under simulated packet loss of 0%, 5% and 15% respectively. The first experiment measured round trip time while the second investigated multiplexing performance. Three different servers were used, one WebSocket and one WebTransport using Socket.IO, and one WebTransport server using the FAILS Component WebTransport module. The results showed that WebTransport reduced Head-of-Line blocking better than WebSocket, especially when the network conditions worsened. It also showed that WebTransport had performance benefits compared to WebSocket [4].

The previous work highlights how different packet sizes can affect performance, loss rates and packet loss in various network protocols. The suggested future work of [4] suggest to investigate if packet size has any impact on performance. How packet loss is affected by packet size in WebSocket and WebTransport protocol is something that has not yet been evaluated. This thesis aims to investigate that question.

3 Background

The following section gives background information on network protocols, the protocols WebSocket and WebTransport, the library Socket.IO, round trip time (RTT), packets, packet size and packet loss.

3.1 Network Protocols

Protocols are used over the internet for all activity that involves communication between two or more entities that are remote from each other [6, p.36]. There are several different protocols used for different reasons. Some of the most common protocols used on the internet is Hypertext Transfer Protocol (HTTP), Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). To understand how a protocol works over the internet one could analyze human analogies [6, p.35]. One could consider a human interaction between two people where one asks the other what time it is. Normally the first person would greet the other, and the other person would greet them back, then comes the question ”What time is it?”, which gives a reply, ”The time is 13:00” [6, p.35-36]. Network protocols works in a similar way, using hardware and software of some device rather than humans [6, p.36]. For example, when trying to access a website through a browser, the computer will send a request message for a connection to the Web server and the server will receive that message and return a reply [6, p.36-37]. When the computer receives this reply, it sends a GET message containing the name of the webpage it is trying to access, and the server will then reply with the webpage to the computer so it can be displayed [6, p.37].

3.2 WebSocket

To combat the many problems HTTP polling caused for applications that needed bidirectional communication between a client and a server, the WebSocket protocol was introduced as an alternative in 2011 as RFC 6455.[1] The protocol works in two parts, the opening handshake and the data transfer. The handshake will happen first, both the client and the server will send their own handshakes, and if they are successful, the data transfer can begin. Since this is a two-way communication, each side can send data at any time they please entirely independent of each other. As the protocol is designed to be an alternative to HTTP polling, it is designed to use existing HTTP infrastructure. Because of this the protocol will work on the HTTP ports 80 and 443 and also support intermediaries and proxies from HTTP.

[1] WebSocket: https://www.rfc-editor.org/rfc/rfc6455.html

3.3 WebTransport

The most recently published draft of the WebTransport Protocol Framework was published in february of 2025.[2] To communicate with a remote server, clients that are constrained by the Web security model can use the WebTransport protocol as it uses a transport that is secure and multiplexed. As head-of-line blocking issues can occur for applications in need of a client to server bidirectional stream of data, WebTransport can be used to combat the issue. The protocol enables multiplexing together multiple streams into a single transport object, thus avoiding the head-of-line blocking. Unreliable datagrams, like UDP, can also be sent using WebTransport.

3.4 Socket.IO

The library Socket.IO provides communication between a client and server where this communication is event based, low-latency and bidirectional.[3] Socket.IO provides different transports from which the connection can be established, HTTP long-polling, WebSocket and WebTransport. In this thesis, WebSocket and WebTransport will be used. A Socket.IO server using WebSocket will not be able to receive a connection from a plain WebSocket client. A plain WebSocket server will not be able to receive a connection from a Socket.IO WebSocket client either. This is because Socket.IO is not a WebSocket implementation.

3.5 Round Trip Time

When sending a packet from a client to a server, one could measure the Round Trip Time (RTT) for that packet by calculating the time it takes for that packet to travel from the client to the server and back to the client [6, p.130]. Included in this time calculation is delays such as packet-queuing delays that occurs in the routers and switches that are intermediate, packet-processing delays and packet-propagation delays [6, p.130]. RTT is also used to estimate the window growth policy in TCP, so a reliable service can be given in an unreliable network [7]. A visual illustration of measuring RTT between a client and a server can be seen in Figure 1.

[2]WebTransport: https://www.ietf.org/archive/id/draft-ietf-webtrans-overview-09.html
[3]Socket.IO: https://socket.io/docs/v4/

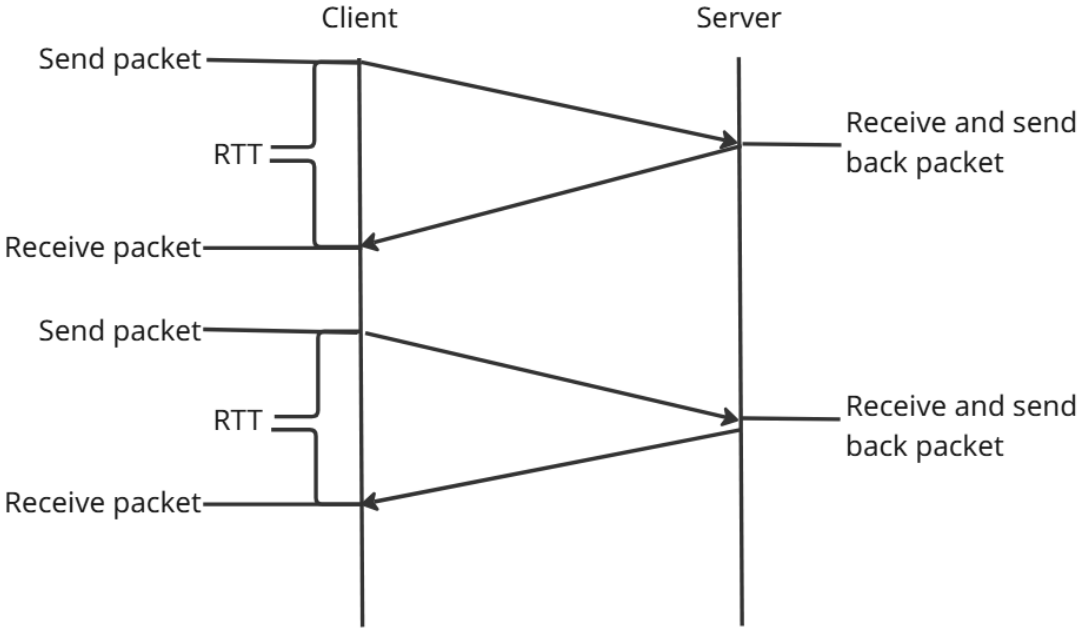

Figure 1: Illustration of calculation of RTT when sending a packet from client to server and back.

3.6 Packets, Packet Size and Packet Loss

A packet is a package of information that contains the data that is to be sent along with header bytes [6, p.32]. These packets can then be sent from a sender to a receiver where the packet is reassembled and the data extracted [6, p.32]. The data in these packets can vary in size, some packets can contain large amounts of data while others contain very little, this is called packet size. Packets sent from sender to receiver run a risk of being dropped, resulting in packet loss. Packet loss is handled differently by different network protocols, TCP ensures packets always arrive to their destination in order, so any dropped packet will be retransmitted, while UDP is unreliable and does no such assurances at all, which could potentially lead to packets being lost [6, p.220]. An example of when packet loss occurs is when a packet arrives at a router where the queue is already full, this will force the router to drop the packet, as there is nowhere else to store it, leading to packet loss [6, p.69]. Having much packet loss in a connection is not good as it can severely impact the performance of the network and lead to delays.

4 Methodology

To investigate the research question of this thesis, several experiments were conducted. All experiments consisted of two servers, one WebSocket and one WebTransport, and a client to each server. Both of these servers and their clients were made using NodeJS and the SocketIO library and their guides for WebSocket[1] and WebTransport[2], allowed by the MIT License.[3] Both servers and their clients have then been modified so the experiments could be run. The code for the experiments can be found on GitHub.[4] All experiments were carried out using a Virutual Machine (VM) and the server and the client were running on the same machine. The specifications of this VM machine can be seen in Table 1.

Table 1 Specification of Viritual Machine used to conduct the experiments.

Operating System	Ubuntu (64-bit)
Host CPU	AMD Ryzen 5 3600
RAM	8GB

As both the client and server were running on the same machine, packet loss had to be introduced to the network, to do this NetworkEmulator (NetEm) was used, which is a tool using Linux traffic control to add packet loss and delays.[5] The packet loss rate simulated was set differently in the experiments and the percentages used were 10%, 15% and 20%.

4.1 Experiment Measuring Round Trip Time (RTT)

This experiment aimed to measure the Round Trip Time (RTT) for a message traveling from the client to the server and back to the client, and was inspired by the experiments conducted in [4] and [8]. The experiment was done in three levels of packet loss, 10%, 15% and 20%. For each level of packet loss four iterations were done sending 1000 messages each, the first iteration having a message length of 1 character, the second 50, the third 100 and the fourth 1000. This difference in message length was done to test different packet sizes. If there is a difference in average RTT between messages 1 characters long and 1000 characters long, that would mean that packet size has an impact on packet loss as that is the only parameter changed. Therefore the average RTT is used as a metric in this experiment.

4.1.1 WebSocket Server and Client

The WebSocket server is launched through a terminal and the client can then connect to the server via a browser. For this experiment, Google Chrome was the browser used. Once the

[1] SocketIO WebSocket tutorial: https://socket.io/docs/v4/tutorial/introduction
[2] SocketIO WebTransport tutorial: https://socket.io/get-started/webtransport
[3] MIT License: https://github.com/socketio/socket.io/blob/main/LICENSE
[4] GitHub with experiment code: https://github.com/CalH71/Exjobb
[5] NetworkEmulator:
https://srtlab.github.io/srt-cookbook/how-to-articles/using-netem-to-emulate-networks.html

client was connected, it waited for 3 seconds before it started sending the messages. When one message is sent, a timer starts, then when the server receives a message from the client, it instantly sends that message back. When the message is returned to the client, the timer is stopped and the RTT for that messages is saved. To ensure proper timings of each message, no new messages were sent until a reply from the previous sent message had been received. When all 1000 messages of one iteration of the experiment has been sent and their RTT measured, the average RTT for all 1000 messages are also calculated and saved.

4.1.2 WebTransport Server and Client

The WebTransport server is also launched through a terminal, but the client is launched through a script that opens a Chromium browser. Once the client is connected to the server it will upgrade the protocol to WebTransport and, just like the WebSocket client, it will wait 3 seconds before it starts to send messages. The messages are sent, received and timed exactly the same as in the WebSocket implementation and the average RTT for all 1000 messages of the iteration are calculated and saved.

5 Results

This section will present the results found in the experiments conducted.

5.1 Websocket

For WebSocket, the experiments showed an increase in average RTT during 10% packet loss where the RTT increased from 40.72ms with a 1 length character message, to 57.09ms with 1000 character length message. During 15% packet loss, the RTT increased from 93.9ms with 1 character length, to 101.27ms with 100 character length, but it then decreased to 92.65ms with 1000 character length. During 20% packet loss there was a decrease from 167.61ms with 1 character length, to 148.3ms with 100 character length, and then an increase with 1000 character length up to 176.14ms. The avarage RTT for each character length message using WebSocket can be found in Figure 2.

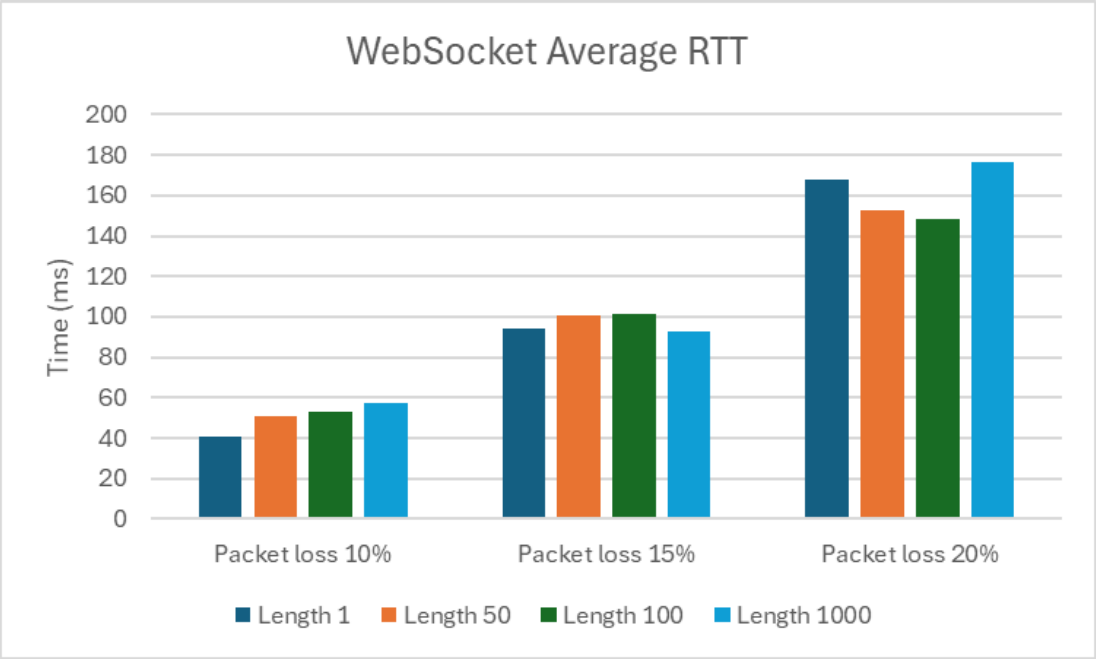

Figure 2: Average RTT for all character lengths in each level of packet loss using WebSocket.

5.2 WebTransport

For WebTransport, the experiments showed a small decrease in average RTT during 10% packet loss from 16.93ms with 1 character length to 14.15ms with 100 character length. With 1000 character length the average RTT increased to 18.04ms. During 15% packet loss the average RTT was 25.47ms with 1 character length, then increase to 27.86ms with 100 character

length, and then decreased to 24.34ms with 1000 character length. During 20% packet loss the largest increase was detected, where 1 character length averaged an RTT of 39.89ms while 1000 character length averaged 59.39ms. The average RTT for each character length using WebTransport can be seen in Figure 3.

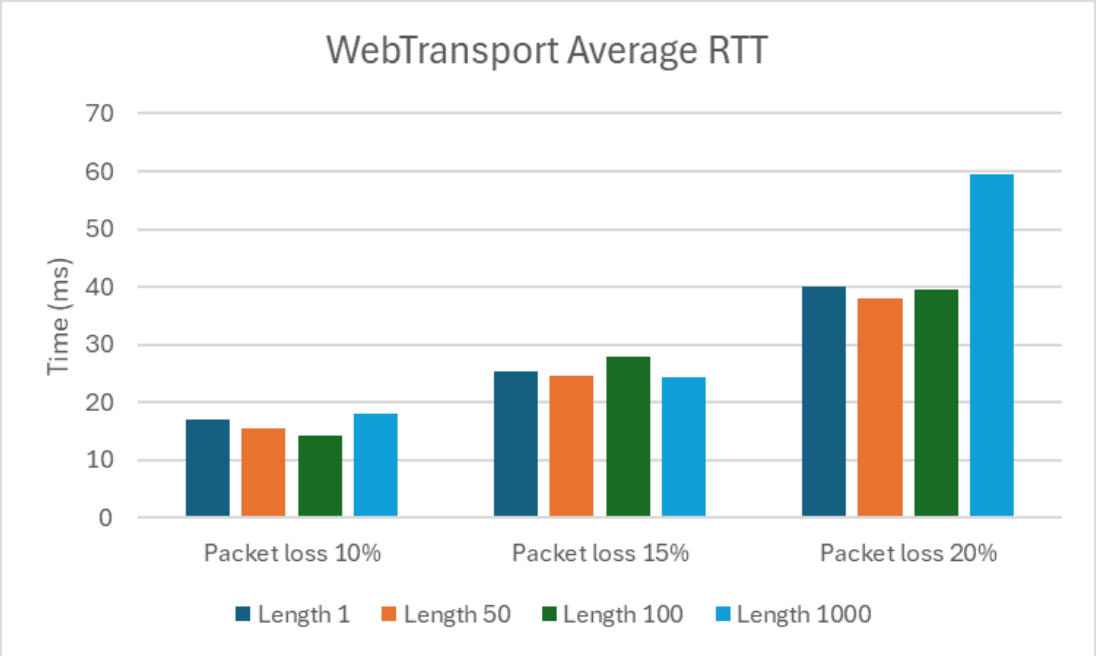

Figure 3: Average RTT for all character lengths in each level of packet loss using WebTransport.

In Table 2 all results from the entire experiment can be found, where the protocol used, the character length used and the average RTT is displayed for each iteration that was done in the experiment.

Table 2 All results from the experiment measuring average RTT for different message sizes.

Packet Loss	Protocol	Character length	Average RTT
10%	WebSocket	1	40,72 ms
10%	WebSocket	50	50,47 ms
10%	WebSocket	100	53,06 ms
10%	WebSocket	1000	57,09 ms
10%	WebTransport	1	16,93 ms
10%	WebTransport	50	15,42 ms
10%	WebTransport	100	14,15 ms
10%	WebTransport	1000	18,04 ms
15%	WebSocket	1	93,90 ms
15%	WebSocket	50	100,87 ms
15%	WebSocket	100	101,27 ms
15%	WebSocket	1000	92,65 ms
15%	WebTransport	1	25,47 ms
15%	WebTransport	50	24,71 ms
15%	WebTransport	100	27,86 ms
15%	WebTransport	1000	24,34 ms
20%	WebSocket	1	167,61 ms
20%	WebSocket	50	152,94 ms
20%	WebSocket	100	148,30 ms
20%	WebSocket	1000	176,14 ms
20%	WebTransport	1	39,89 ms
20%	WebTransport	50	38,02 ms
20%	WebTransport	100	39,51 ms
20%	WebTransport	1000	59,39 ms

6 Discussion

The goal of this thesis was to investigate if packet size had any impact on packet loss in the protocols WebSocket and WebTransport. This section will evaluate the results found and answer the research question of this thesis, as well as describe some limitations.

6.1 Analysis

In the presented results small increases and decreases in average RTT was recorded for both WebSocket and WebTransport in all three levels of packet loss, which can be seen in Figure 2, Figure 3 and Table 2. These increases and decreases were however relatively small and showed no clear pattern. For example, in WebSocket, during 10% packet loss the average RTT showed an increase as the packet size grew, but during 15% and 20% decreases were detected as packet size grew, as seen in Figure 2 and Table 2. In WebTransport the average RTT both increases and decreases as the packet size grows in all levels of packet loss, as seen in Figure 3 and Table 2, but the differences are relatively small. There are two instances of a quite significant increase in the results. The first one being in WebSocket 10% packet loss, where the average RTT increases from 40.72ms with 1 character length to 57.09ms with 1000 character length, which can be seen in Figure 2 and Table 2. The second one is in WebTransport 20% packet loss where the average RTT jumped from 38-39ms with 1-100 character length, to 59.39ms with 1000 character length, as seen in Figure 3 and Table 2. Given that no other results showed such a steep increase (or decrease) compared to other character lengths, the cause is most likely that some packets were dropped several times in a row (not as an effect of the increase in packet size), leading to a high RTT spiking the average. Despite these two big increases, when analyzing all results combined there is no clear pattern to be seen.

The results produced suggest that a reason packet size does not affect packet loss in WebSocket and WebTransport is that larger packets are just as likely to experience packet loss and be dropped as small packets are. In the experimental setup, the network has a set percentage of packet loss and the packets that are dropped are not handpicked but rather dropped randomly, thus larger packets are just as likely as smaller packets to get dropped. Another reason for the result could be that the environment of the network is equally beneficial for both small and larger packet sizes. For example, in [1] it was found that larger packet sizes benefit the performance of TCP-Reno when the connection is dominated by link errors. And when the connection is dominated by congestion instead, smaller packet sizes are more beneficial. Considering this, the results might suggest that the test environment and the connection used in the experiment is equally beneficial for both larger and smaller packet sizes.

Considering these results, it is evident that packet size does not affect packet loss in WebSocket and WebTransport and there is no benefit to be gained using smaller packet sizes compared to larger and vice versa.

Just like the results found in [4], the results here also showed that WebTransport performs significantly better than WebSocket in average RTT along all tested levels of packet loss. In 20% packet loss, the difference in average RTT was 128ms in favor of WebTransport, which can be seen in Table 2.

6.2 Limitations

The experiments conducted measured RTT and then manipulated the character length of messages to evaluate the effect of packet size on packet loss. This could be seen as a limitation as it would be more accurate to measure the total number of packets that were dropped out of the ones that were sent. For this thesis, no effective way of measuring could be done.

Another limitation is that the experiments were done where both the client and server were run on a single machine. Just like [4] mentions in the limitations, real-world scenarios might have challenges that are not presented here as the client and server are not remote from each other.

The use of Netem to simulate packet loss in the network can also be seen as a limitation. As real life packet loss circumstances might provide different challenges that are not captured in this thesis.

Despite these limitations, the results and analysis of this thesis should still be considered valuable as the experiments and analysis provide an insightful evaluation of the research question which can be used for future research within the field.

7 Conclusion

This thesis has investigated if packet size has any impact on packet loss in the protocols WebSocket and WebTransport. As discussed in Section 6.1, the experiments conducted in this thesis showed no clear pattern that larger or smaller packet sizes have an impact on packet loss, and that neither are beneficial in that regard. This is likely because larger and smaller packet sizes have the same probability of experiencing packet loss in a connection that has a set percentage of packet loss, as well as the environment and connection are equally beneficial for both larger and smaller packets. Considering these results, developers using Socket.IO with WebSocket and WebTransport can take the findings of this thesis into account when using different packet sizes in packet loss heavy environments to ensure packets are rarely dropped.

7.1 Future/Suggested Work

Future work in this field could investigate more precise measures than measuring RTT for messages sent. One could investigate how many packets were dropped compared to how many were sent using different packet sizes. Future work could also expand upon this research and investigate other network protocols to see if the results will differ from the ones found in this thesis.

References

[1] M. Albuquerque, J.H. Kim, and S. Roy. Effect of packet size on tcp-reno performance over lossy, congested links. In *2001 MILCOM Proceedings Communications for Network-Centric Operations: Creating the Information Force (Cat. No.01CH37277)*, volume 1, pages 705–710 vol.1, 2001.

[2] J. M Boyce and R. D Gaglianello. Packet loss effects on mpeg video sent over the public internet. In *Proceedings of the sixth ACM international conference on Multimedia*, pages 181–190, 1998.

[3] G. Dán, V. Fodor, and G. Karlsson. Packet size distribution: An aside? In Marco Ajmone Marsan, Giuseppe Bianchi, Marco Listanti, and Michela Meo, editors, *Quality of Service in Multiservice IP Networks*, pages 75–87, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[4] C. Gulliksson. Empirical analysis of the impact of packet loss on webtransport using socket.io. *Bachelor thesis, Umeå University*, 2024.

[5] J. Korhonen and Y. Wang. Effect of packet size on loss rate and delay in wireless links. In *IEEE Wireless Communications and Networking Conference, 2005*, volume 3, pages 1608–1613 Vol. 3, 2005.

[6] J. F. Kurose and K. W. Ross. *Computer Networking A Top-Down Approach*. Pearson, seventh edition, 2017.

[7] D. S. Kushwah, M. Kumar, and L. P. Verma. Analyzing reliable transport layer protocol performance. *Journal of Physics: Conference Series*, 1714(1):012040, jan 2021.

[8] D. Williamson and R. O'Reilly. Webtransport and websockets: An empirical analysis of connection time, message response, and payload efficiency. In *2023 34th Irish Signals and Systems Conference (ISSC)*, pages 1–6, 2023.

UMEÅ UNIVERSITY