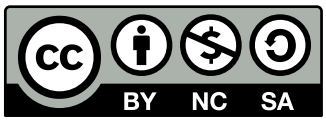
Aalto University
School of Electrical
Engineering

Master's Programme in Computer, Communication and Information Sciences

Performance of Server Message Block implementations over QUIC

David Enberg

Master's Thesis
2025



Aalto University
School of Electrical
Engineering

Author David Enberg

Title Performance of Server Message Block implementations over QUIC

Degree programme Computer, Communication and Information Sciences

Major Communications Engineering

Supervisor PhD Pasi Sarolahti

Advisor Bastian Shajit (MSc)

Collaborative partner Tuxera Oy

Date 28 November 2025 **Number of pages** 18 **Language** English

Abstract

Keywords For keywords choose, concepts that are, central to your, thesis

Aalto University
School of Electrical
Engineering

Författare David Enberg

Titel Arbetets titel

Utbildningsprogram Electronik och electroteknik

Huvudämne Communications Engineering

Övervakare Prof. Pirjo Professori

Handledare TkD Alan Advisor, DI Elsa Expert

Samarbetspartner Company or institute name in Swedish (if relevant)

Datum 28 November 2025 **Sidantal** 18 **Språk** engelska

Sammandrag

Nyckelord Nyckelord på svenska, temperatur

Preface

Otaniemi, 30 June 2025

Eddie E. Engineer

Contents

Abbreviations

ACK	acknowledgment
IP	Internet Protocol
ISN	Initial Sequence Number
RFC	Request For Comment
SMB	Server Message Block
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

1 Introduction

1.1 Research questions and objectives

1.2 Thesis structure

2 Background

2.1 Internet transport protocols

2.1.1 Transmission Control Protocol

The Transmission Control Protocol (TCP), as outlined in Request For Comment (RFC) 793 is a foundational internet transport protocol. It was originally published in September 1981, focusing primarily on solving military communication challenges. It is intended to be a highly reliable transport protocol between hosts in a packet switched network. The TCP is connection-oriented, providing reliable, end-to-end, bi-directional communication between a pair of processes, in the form of a continuous stream of bytes. The TCP protocol is designed to fit into a layered hierarchy of protocols, slotting in on top of the internet protocol (IP)[1]. IP handles the addressing and routing of datagrams between the hosts, while TCP aims to ensure that information is delivered correctly, in order, and without duplications, without any reliability guarantees needed from the underlying protocol, which may lose, fragment or reorder the datagrams[2].

TCP ensures reliable communication by using a system of sequence numbers and acknowledgments (ACKs). Each transmitted byte of data is assigned a sequence number, and the peer is required to send an ACK to acknowledge that the data was received. On the receiver side the sequence numbers are used to reconstruct the data, ensuring that the data is received in order. If the sender does not received an ACK within a timeout period, the missing segment will be retransmitted. A checksum is included with each segment, ensuring that the datagram was not corrupted during transport. If data corruption is detected, the receiver will discard the damaged segment, and rely on the retransmission mechanic to recover. The TCP uses a receive window for flow control, allowing the receiver to decide the amount of data that the sender may send before waiting for further ACKs. The reliability and flow control aspects of the TCP demands that the TCP store some information about the transmission. The data stored about the data stream, sockets, sequence numbers and windows sizes, is referred to as a connection. The network address and port tuple is referred to as a socket, and a pair of sockets is used in identifying the connection. Using this mechanic to uniquely identify connections, allows for multiple processes to simultaneously communicate using the TCP[2].

The TCP header, figure 1, encodes the functionality of the TCP. It follows the IP header in a datagram. The header is usually 20 bytes long, but can be extended using options. It begins with the source and destination port, which together with the source and destination addresses, are used to identify the connection. The next two fields in the header are the sequence and acknowledgement numbers. The sequence number is the sequence number of the first data byte in the data segment. If the SYN flag is set the sequence number is referring to the initial sequence number (ISN). The acknowledgement refers to the next sequence number the receiver is expecting to receive, at the same time acknowledging that all sequence numbers up to this point was received. Next is the data offset field, indicating where the data begins. The reserved

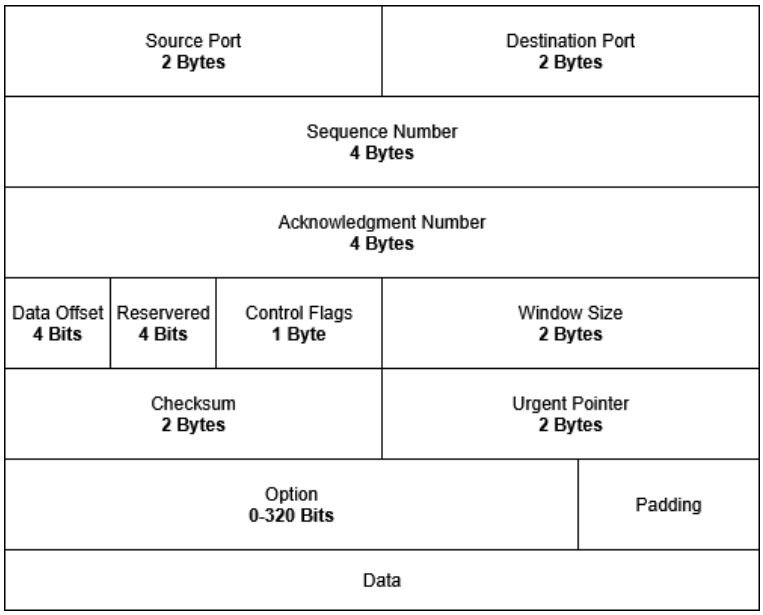


Figure 1: The TCP header

field following this must be 0. After this is the 1 byte flags field

- **URG** Urgent pointer field is set
- **ACK** acknowledgement field is set
- **PSH** Push function, requesting that buffered data is sent immediately to the receiver
- **RST** Reset the connection
- **SYN** Synchronize sequence numbers
- **FIN** Sender is done sending data

The 2 byte window field specifies the number of bytes that may be in-flight at any one time. This is the specified size of the sliding window that is used for flow control purposes. Following the window field is a 2 byte checksum field, used for detecting corruption of the TCP-header, data payload as well as a pseudo IP header, containing information about the source and destination addresses, as well as the protocol number and tcp packet length. In case the URG bit is set in the flags field, the 2 byte urgent pointer header field indicates where the urgent data ends. Finally the options field contains extension to the normal TCP header, containing among other, options for maximum segment size and multipath TCP[3].

To ensure reliable delivery of TCP segments, each segment is assigned a sequence number. This allows the receiver to reconstruct segments delivered out of order, and additionally detect missing segments. The receiver sends acknowledgments, containing the next expected sequence number, to signal to the sender that the data

was successfully received. The sequence number is a 32 bit number, with the initial sequence number (ISN) selected randomly at the time when the TCP connection is established. This ensures that sequence numbers from stale connections have a low probability of overlapping with any active connection[2].

The TCP connection is established via a three-way handshake. The client sends a TCP packet with the SYN bit set in the flags field, this packet contains the clients ISN. The server responds with a packet with the SYN and ACK bit set, acknowledging the clients sequence number as well as providing the servers ISN. Finally the client responds with an ACK, acknowledging the servers ISN. Following this the client and server are synchronized, and communication may begin. A peer may terminate its side of the connection by sending a FIN packet, signaling to the other endpoint that one side has closed its side of the communication. The closed endpoint may continue receiving data until the other endpoint also closes it side[2].

2.1.2 User Datagram Protocol

The User Datagram Protocol (UDP), which was defined by RFC 768, is designed to enable programs to transmit self-contained messages, know as datagrams, over a packet-switched network. The UDP is designed to run on top of the IP[1], using IP addresses and port numbers for addressing. The UDP is by design connectionless, providing no guarantees for datagram delivery, duplicate datagrams or in-order delivery. In exchange, the UDP aims to minimize the overhead present in the protocol. As UDP is connectionless there is no need to establish a connection via a handshake, instead datagrams can be transmitted directly, and they should be designed in such a way that they can stand on their own. The UDP header, as seen in figure 2 is only 8 bytes long, consisting of the source and destination port, as well as the length of the datagram and a checksum to verify the received datagram[4]. Even though UDP has a checksum field its use varies depending on the implementation. Some implementations may discard the datagram, or alternatively pass it along to the application with a warning, as UDP provides no way to recover from broken datagrams[5]. The minimal UDP header (8 bytes) combined with the lack of a handshake makes UDP a protocol with the bare minimum needed for datagram transfer. Generally, UDP is used for applications where low latency is a requirement and some amount of packet loss is deemed acceptable. It is then up to the application layer to handle missing, reordered or duplicate datagrams.

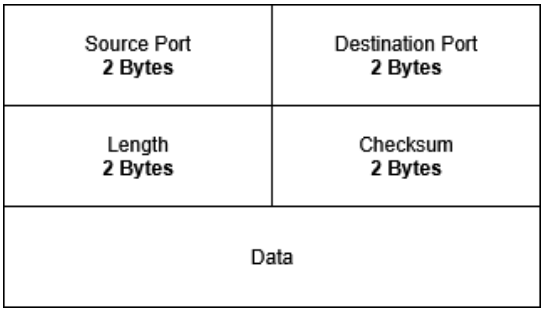


Figure 2: The UDP header

2.2 The SMB protocol

3 QUIC

3.1 Information about the QUIC protocol

4 The SMB protocol

4.1 Information about the SMB protocol

5 Implementing QUIC as transport for SMB server

5.1 MsQuic architecture and API

5.2 Fusion SMB server QUIC transport layer design

6 Performance and interoperability benchmarking

6.1 Tesn environment

6.1.1 Hardware environment

6.1.2 SMB over QUIC implementations analyzed

Windows SMB client/server

Fusion SMB server

6.2 Test scenarios

6.2.1 interoperability tests

6.2.2 Becnhmarking workloads

6.3 Results

7 Conclusions

7.1 Discussion

7.2 Future work

References

[1] *Internet Protocol*. RFC 791. Sept. 1981. DOI: 10.17487/RFC0791. URL: https://www.rfc-editor.org/info/rfc791.

[2] *Transmission Control Protocol*. RFC 793. Sept. 1981. DOI: 10.17487/RFC0793. URL: https://www.rfc-editor.org/info/rfc793.

[3] A. Ford et al. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 8684. Mar. 2020. DOI: 10.17487/RFC8684. URL: https://www.rfc-editor.org/info/rfc8684.

[4] *User Datagram Protocol*. RFC 768. Aug. 1980. DOI: 10.17487/RFC0768. URL: https://www.rfc-editor.org/info/rfc768.

[5] J. Kurose and K. Ross. *Computer networking: A top-down approach, global edition*. en. 8th ed. London, England: Pearson Education, June 2021.