User Space QUIC: Prototype and Performance Study

Swapnil Garg	22115150
Mmukul Khedekar	22114054

Department of Computer Science and Engineering
Indian Institute of Technology Roorkee
Roorkee - 247667 (INDIA)

Supervisor : Dr. Manoj Misra

November, 2025

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE, ROORKEE

Candidates' Declaration

We hereby certify that the work presented in this report entitled "User Space QUIC: Prototype and Performance Study" in partial fulfillment of the requirements for the award of the degree of Bachelor Of Technology and submitted in the Department of Computer Science and Engineering of the Indian Institute of Technology Roorkee is an authentic record of our own work carried out during the period from August, 2025 to November, 2025 under the supervision of Dr. Manoj Misra, Professor.

Swapnil

Swapnil Garg
(22115150)

Mmukul

Mmukul Khedekar
(22114054)

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Manoj
Digitally signed by Manoj
DN: cn=Manoj, o=IIT, ou, email=manojfec@gmail.com, c=IN
Date: 2025.12.03 07:36:46 +05'30'

Date: November, 2025

Supervisor
(**Dr. Manoj Misra**)

Acknowledgement

We express our sincere gratitude to our supervisor, Dr. Manoj Misra, for his guidance, constant encouragement, and intellectual support throughout the duration of this project. His expertise and insightful critiques were invaluable in shaping this work. We also thank the faculty and staff of the Department of Computer Science and Engineering for providing us with the necessary resources and environment for research.

Abstract

The Quick UDP Internet Connections (QUIC) protocol is a modern transport layer protocol designed by Google to supersede the limitations of TCP combined with TLS. This project focuses on implementing a user-space prototype of the QUIC protocol and conducting a performance comparison against the traditional TCP+TLS stack. We investigate QUIC's unique selling points (USPs) such as reduced connection establishment latency via 1-RTT and 0-RTT handshakes, integrated stream multiplexing to eliminate Head-of-Line (HoL) blocking, and built-in security without the overhead of TLS records. Our experimental setup, involving both real-world and controlled network environments (using `netem` and `netns`), demonstrates that QUIC significantly outperforms TCP+TLS, particularly in scenarios involving connection reestablishment and packet loss. This user-space implementation serves as a prototype for exploring a full kernel-level implementation in future work.

Contents

Chapter 1

Introduction

1.1 QUIC: Quick UDP Internet Connections

QUIC is a modern, secure, and multiplexed Transport Layer protocol built on top of UDP. It was developed to overcome long-standing limitations of TCP, such as, slow connection setup, Head-of-Line blocking, the overhead of integrating a separate TLS layer, etc. QUIC provides built-in encryption using TLS 1.3, supports connection migration across networks, and enables independent, non-blocking streams within a single connection.

It was originally designed by Google in 2013 and later standardized by the IETF in 2021 as the transport layer protocol for HTTP/3 and is rapidly becoming the default protocol for web communication.

This project uses a user-space implementation of QUIC to study its behavior, evaluate its performance characteristics, and explore its feasibility for future kernel-level integration.

1.2 Problem Statement

The central problem addressed by this project is the performance bottleneck inherent in the TCP+TLS stack, particularly related to connection establishment latency, Head-of-Line (HoL) blocking, and encryption overhead. The objective is:

1. To investigate and quantify why QUIC performs better than TCP.
2. To implement a functional QUIC prototype in user space to empirically demonstrate its advantages over the standard TCP+TLS stack.
3. To compare the latency, throughput, and connection establishment time of our QUIC implementation against TCP and TCP+TLS under various network conditions.

1.3 Literature Review

In this section, we examine the key areas in which QUIC introduces optimizations over the traditional TCP+TLS stack, as documented in prior research and standards literature.

1.3.1 Traditional Transport Layer Limitations (TCP+TLS)

The standard web stack relies on TCP for reliability and TLS for security. The major limitations reviewed include:

- **Head-of-Line (HoL) Blocking**[4]**:** If a single TCP segment is lost, the entire pipeline of streams waiting on that TCP connection must stall until the segment is retransmitted and received.
- **High Latency Handshakes:** Establishing a TCP connection requires a 3-way handshake, followed by an additional 1-RTT (or more) for the TLS 1.3 handshake, leading to significant initial connection latency.
- **TLS Overhead**[5]**:** Encapsulating application data within TLS records adds headers, authentication tags, and internal buffering, incurring CPU and memory overhead.

1.3.2 QUIC Design

The literature confirms that QUIC was designed to address these fundamental issues:

- **Stream Multiplexing**[4]**:** By running streams independently over UDP, QUIC eliminates HoL blocking, as packet loss in one stream does not affect others.
- **Reduced Handshake Latency:** QUIC incorporates a symmetric key exchange and provides fully integrated encryption, often achieving a 1-RTT connection establishment. For resumed connections, it supports **0-RTT connection establishment** by utilizing cached server-side parameters.
- **Integrated Security**[7]**:** QUIC provides security without the full TLS overhead by applying encryption directly to the packets, avoiding TLS records, headers, and internal buffers.
- **Connection Migration:** Since QUIC connections are identified by a Connection ID (not a 4-tuple like TCP), connections can persist across changes in the user's IP address (e.g., switching from Wi-Fi to cellular).

Chapter 2

Work Done

The primary work completed involves the design, implementation, and rigorous testing of the user-space QUIC prototype. We followed IETF standards (RFC 8999[6], 9000[3], 9001[7] and 9002[2]) for reference on packet structure, handshake sequence, packet encryption, connection management etc.

2.1 Implementation Details

The core implementation involved building QUIC's unique features on top of the UDP socket layer:

2.1.1 Connection Handshake and Encryption

Unlike the TCP+TLS Handshake (which involves a 3-way TCP handshake followed by a 2-message TLS 1.3 handshake), the QUIC implementation provides a faster sequence[3]:

- **1-RTT Connection:** The Client sends a `ClientHello` and the Server responds with `ServerHello`, completing the cryptographic and transport handshake within one RTT.
- **0-RTT Connection Establishment:** For clients re-connecting to a server, QUIC utilizes cached server parameters to send application data immediately with the first packet, achieving a near-zero latency connection establishment (0 ms RTT, as shown in the results).

2.1.2 QUIC Error Control

The prototype utilizes Acknowledgement-based Loss Detection:

- **ACK-based Reliability:** QUIC Acknowledgements (ACKs) ensure packets are received.

- **Loss Detection Thresholds:** A packet is declared lost if it remains unacknowledged after either a **Time Threshold** (in-flight packet is unacknowledged for more than t seconds) or a **Packet Threshold** (an unacknowledged packet was sent k packets before an acknowledged packet).

2.2 Experimental Setup and Results

Two testing environments were established to capture comprehensive statistics:

1. **IITR-WIFI:** Used to observe protocol behavior under real-world, unpredictable network conditions.
2. **Ubuntu (`netem` + `netns`):** A controllable testing environment where parameters like packet loss and delay could be artificially and precisely introduced for repeatable testing.[1]

We used OpenSSL 3.0 as the standard implementation of TLS 1.3 and Linux Socket API as the standard implementation of UDP and TCP to test against our implementation of QUIC.

Testbench

The testing was done on Asus ROG Zephyrus G15 of the following specifications-

1. **CPU**: AMD Ryzen 9 6900HS
2. **RAM**: 16GB
3. **Network Card**: MediaTek Wi-Fi 6E MT7992 Wireless Lan
4. **Operating System**: Ubuntu 24.04 LTS

2.2.1 Latency

We measured the network latency when sending 500 packets of varying sizes.

We then measured the same statistics while varying the packet count and keeping the packet size constant (1024 bytes per packet).

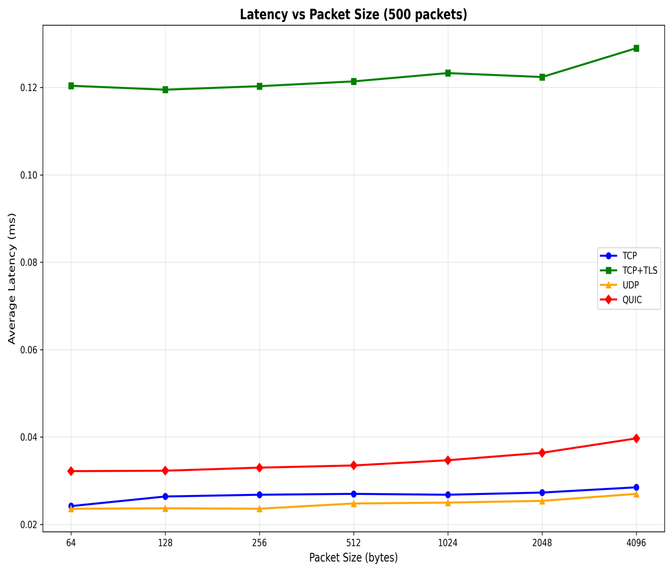

Figure 2.1: Comparison of Average Latency (ms) vs. Packet size (bytes)

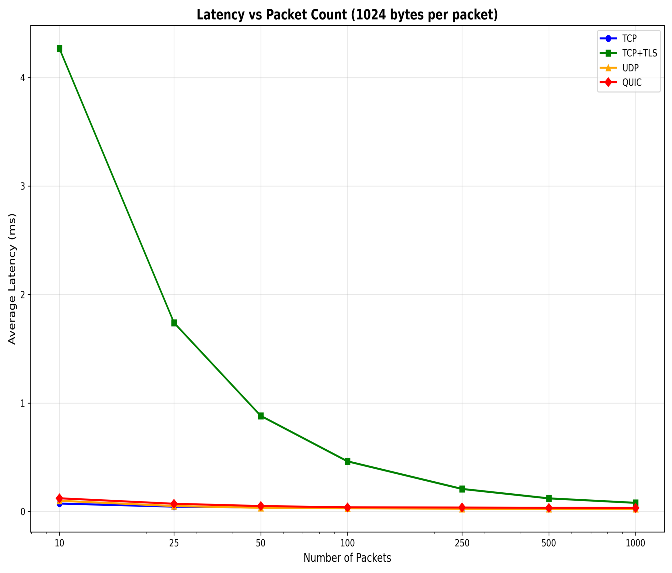

Figure 2.2: Comparison of Average Latency (ms) vs. Packet Count

2.2.2 Throughput

We measured the average throughput when sending 500 packets of varying sizes.

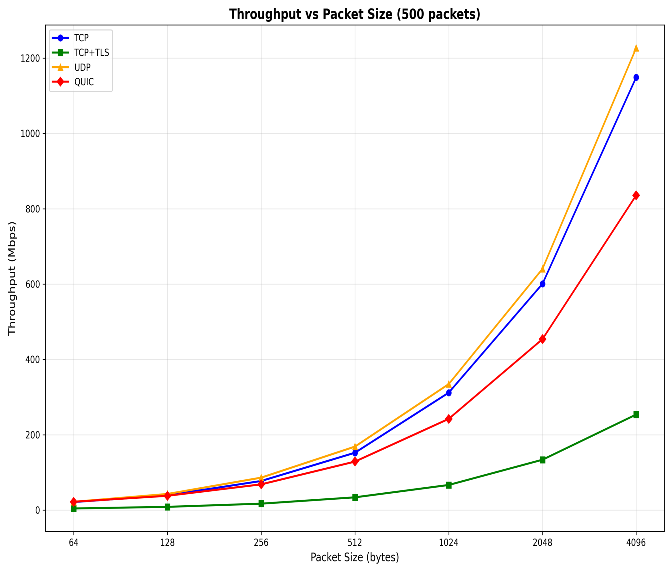

Figure 2.3: Comparison of Average Throughput (Mbps) vs. Packet size (bytes)

2.2.3 Handshake Time

We aimed to measure the performance of the QUIC handshake in comparison to the handshakes of other transport-layer protocols. To do this, we measured the handshake time while sending packets of varying sizes.

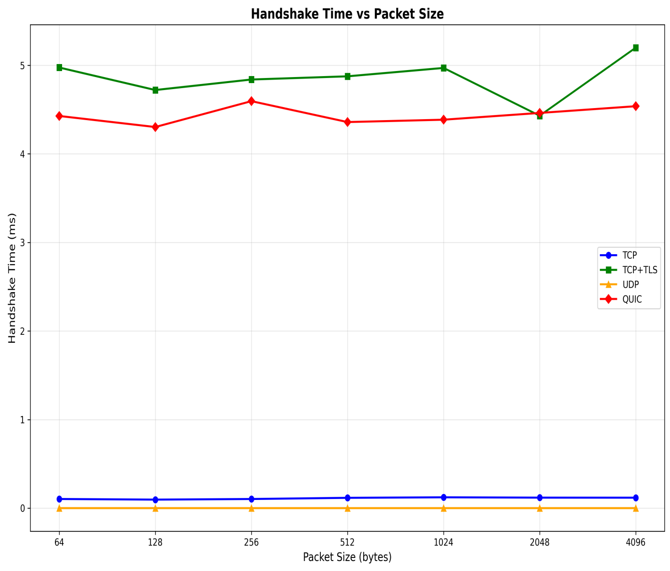

Figure 2.4: Comparison of Handshake Time (ms) vs. Packet size (bytes)

2.2.4 Reconnection Latency

We then evaluated the latency achieved when a client reconnects to a server. For this, we sent packets across varying numbers of reconnection attempts.

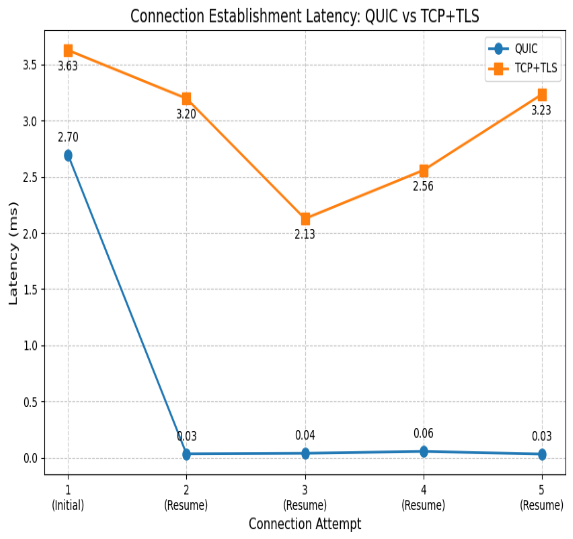

Figure 2.5: Comparison of Reconnection Latency (ms) vs. Number of reconnects

2.2.5 Packet Loss

To measure latency and throughput under consistent packet-loss conditions, we used a network emulator and evaluated our implementation in a controlled simulation environment with varying loss rates.

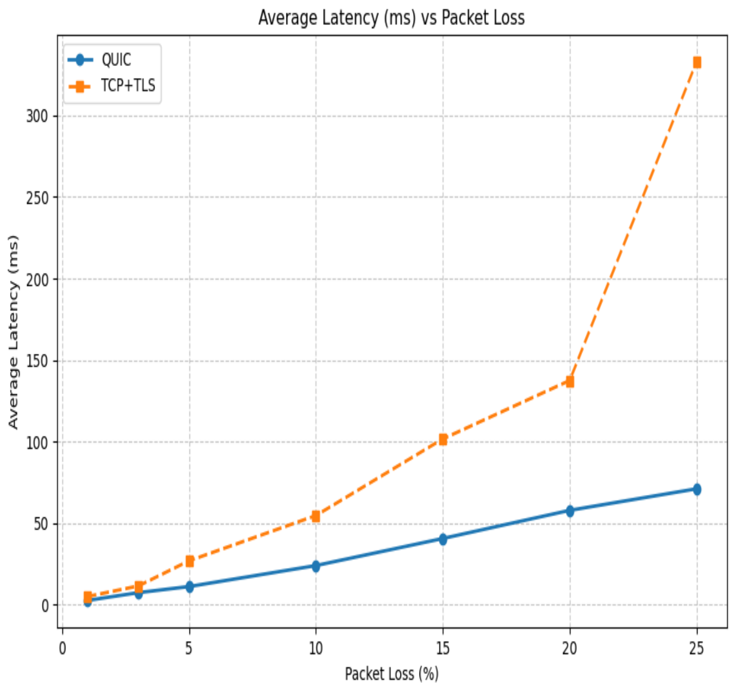

Figure 2.6: Comparison of Latency (ms) vs. Packet Loss

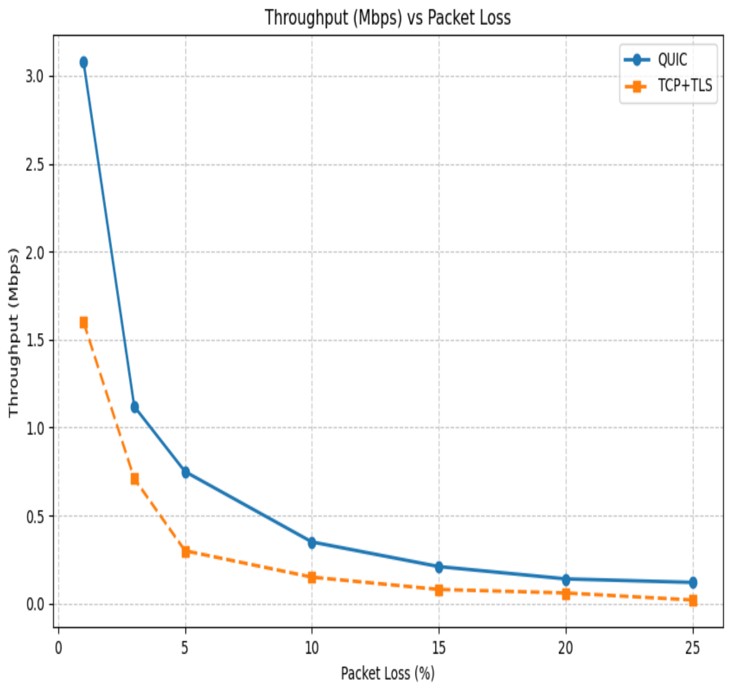

Figure 2.7: Comparison of Throughput (Mbps) vs. Packet Loss

Chapter 3

Conclusion and Future Work

3.1 Conclusion

This project successfully developed and tested a user-space QUIC prototype. The empirical results verify the key Unique Selling Points (USPs) of QUIC: significantly faster connection resumption via 0-RTT, reduced latency, and improved throughput stability under packet loss compared to the TCP+TLS stack. This prototype serves as a proof-of-concept for our future work to be done.

3.2 Work to be Done (Future Work)

The next phase of this project will focus on transitioning to a kernel-level implementation:

1. **Implementing Congestion Control:** Implementing the Congestion Control algorithm documented for QUIC optimize data transmission rates.
2. **Learning Kernel Programming:** Acquiring the necessary skills for low-level system development.
3. **Linux Kernel Implementation:** Developing a native Linux Kernel level implementation of QUIC.
4. **Final Testing:** Conducting a comprehensive performance comparison of the kernel-level QUIC implementation against the standard Linux TCP stack.

Bibliography

[1] Stephen Hemminger. “Network Emulation with NetEm”. In: *Linux Conference Australia (LCA)*. Canberra, Australia, Apr. 2005.

[2] J. Iyengar and I. Swett. *QUIC Loss Detection and Congestion Control*. RFC 9002. May 2021. DOI: 10.17487/RFC9002. URL: https://www.rfc-editor.org/info/rfc9002.

[3] J. Iyengar and M. Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. May 2021. DOI: 10.17487/RFC9000. URL: https://www.rfc-editor.org/info/rfc9000.

[4] Adam Langley et al. “The QUIC Transport Protocol: Design and Internet-Scale Deployment”. In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM ’17. Los Angeles, CA, USA: Association for Computing Machinery, 2017, pp. 183–196. ISBN: 9781450346535. DOI: 10.1145/3098822.3098842. URL: https://doi.org/10.1145/3098822.3098842.

[5] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/info/rfc8446.

[6] M. Thomson. *Version-Independent Properties of QUIC*. RFC 8999. May 2021. DOI: 10.17487/RFC8999. URL: https://www.rfc-editor.org/info/rfc8999.

[7] M. Thomson and S. Turner. *Using TLS to Secure QUIC*. RFC 9001. May 2021. DOI: 10.17487/RFC9001. URL: https://www.rfc-editor.org/info/rfc9001.