BTP Project Proposal: A Comparative Study of QUIC at the Kernel Level

Swapnil Garg
22115150

Mmukul Khedekar
22114054

September 22, 2025

Abstract

This report outlines a project proposal for a comparative study of the QUIC transport protocol implemented at different layers of the network stack. The project aims to compare the performance of application-level QUIC, which is the current standard, with a custom-built kernel-level QUIC implementation. The performance of both QUIC implementations will be benchmarked against the traditional TCP transport protocol. The study will focus on key performance metrics such as throughput, latency, CPU utilization, and packet loss recovery. The ultimate goal is to determine if a kernel-level implementation of QUIC can offer significant performance advantages over its user-space counterpart such as by reducing context switching overhead and as well as investigate other areas of optimisations and improvements. This proposal also details the project timeline, methodology, and a review of existing literature.

Contents

1 Introduction

The rise of modern web applications and real-time services has created a demand for transport protocols that are more efficient and secure than the traditional TCP/IP stack. The QUIC (Quick UDP Internet Connections) protocol, originally developed by Google, has emerged as a promising solution. Built on top of UDP, QUIC provides several key improvements, including reduced connection establishment latency, stream multiplexing to prevent head-of-line blocking, and integrated TLS 1.3 encryption.

While QUIC has gained widespread adoption, almost all production-ready implementations exist in the application user space. This user-space deployment allows for rapid development and deployment cycles, as updates do not require modifications to the operating system kernel. However, this approach can introduce performance overhead due to repeated context switches and data copying between the user and kernel spaces.

This project proposes to explore the potential benefits of a kernel-level QUIC implementation. By moving the protocol logic directly into the kernel, we hypothesize that it will be possible to reduce the processing overhead and achieve a more streamlined data path, leading to superior performance in comparison to both TCP and application-level QUIC.

2 Literature Review

QUIC has been the subject of extensive research since its inception. Early studies by Google and others highlighted its benefits over TCP, particularly in reducing web page load times and improving video streaming performance. Key findings from existing literature include:

- **Performance in Lossy Networks:** A number of studies, such as those comparing QUIC and TCP over LTE networks using 'ns-3' simulations, have demonstrated QUIC's superior throughput and connection establishment times in environments with high packet loss and latency.
- **Head-of-Line Blocking:** The native stream multiplexing of QUIC has been confirmed to effectively mitigate HoL blocking, which is a major limitation of TCP's single, ordered byte stream.
- **CPU Overhead:** Research has consistently identified higher CPU utilization for user-space QUIC implementations compared to in-kernel TCP, especially at high bandwidths. This is primarily attributed to the lack of kernel-level optimizations (e.g., TCP Segmentation Offload) for UDP flows and the overhead of processing protocol logic in user space.
- **Kernel-Level QUIC:** The concept of an in-kernel QUIC implementation has been explored in early patch sets for the Linux kernel. These initial tests showed that while an in-kernel QUIC implementation can be successfully implemented, it may not immediately outperform a highly-optimized in-kernel TCP stack, indicating a need for further optimization.

Our project seeks to build upon this existing research by providing a direct, hands-on comparison of the performance trade-offs between the two most common deployment models for transport protocols.

3 Project Proposal and Methodology

Our project will involve three main phases to accomplish the comparative study.

3.1 Phase 1: Application-level QUIC and TCP Baseline

The first phase will focus on implementing a simple client-server model using a popular, well-maintained application-level QUIC library. This implementation will serve as one of the baselines for our comparison. We will use this setup to collect initial performance data for both application-level QUIC and a standard TCP connection under controlled network conditions. The primary goal of this phase is to become proficient with the QUIC protocol specification and to establish a reliable testing environment.

3.2 Phase 2: Kernel-level Implementation and Learning

During the winter break, we will dedicate time to understanding and learning Linux kernel programming. This is the most crucial part of the project. We will focus on the following:

- Understanding the Linux network stack architecture.
- Writing and testing basic kernel modules.
- Designing the architecture of our kernel-level QUIC implementation to handle core protocol features such as packet parsing, stream management, and congestion control.

3.3 Phase 3: Comparative Analysis and Final Report

In the final phase, we will implement the designed kernel-level QUIC stack. Once the implementation is complete and functional, we will conduct a series of controlled experiments to compare the three protocols: kernel-level QUIC, application-level QUIC, and TCP.

3.3.1 Key Performance Metrics

The comparison will be quantitative and based on the following metrics:

- **Throughput:** Measured as the data transfer rate in megabits per second (Mbps). We will conduct tests under varying bandwidth limits.
- **Latency:** Measured as Round-Trip Time (RTT) for data packets and total time for connection establishment. We will introduce artificial network delays to observe performance under different conditions.
- **CPU Utilization:** Measured by monitoring CPU usage of the server process to quantify the overhead of each protocol stack.
- **Packet Loss Recovery:** Measured by introducing controlled packet loss and observing the time taken to recover and the impact on overall throughput.
- **Jitter:** Measured as the variation in packet delay to understand the suitability of each protocol for real-time applications.

4 Timeline and Deliverables

Phase	Tasks & Deliverables	Timeline
Phase 1 (Current Semester)	**Application-level Implementation** • Study the QUIC protocol specification • Set up a local test environment • Implement client-server application using user-space QUIC library • Conduct initial performance tests for baseline data • **Deliverable:** Working user-space QUIC prototype + performance data	**Ongoing**
Phase 2 (Winter Break)	**Kernel Programming and Design** • Learn Linux kernel programming fundamentals • Study Linux network stack and kernel module development • Design architecture for kernel-level QUIC implementation • **Deliverable:** Detailed design document for kernel-level stack	**Nov-Dec**
Phase 3 **(Next Semester)**	**Kernel-level Implementation and Evaluation** • Implement kernel-level QUIC stack as a module • Conduct comprehensive performance tests on all protocols • Analyze collected data and interpret results • Write final project report with findings • Prepare final presentation • **Deliverable:** Working kernel-level QUIC prototype + final report	**Jan-Apr**

5 Conclusion and Future Work

This project aims to provide a definitive comparison between the performance characteristics of QUIC implemented in user space versus the kernel, using TCP as a well-established benchmark. The findings from this study will contribute to the ongoing discussion about the optimal deployment for modern transport protocols. Future work could include implementing more advanced QUIC features like multipath support or exploring hardware offloading capabilities for a kernel-level implementation.