Performance Comparison of QUIC and TCL+TLS Protocols in a File Transfer Application

1 Introduction

In a time dominated by data-driven applications, efficient and secure file transfer protocols are critical for enabling real-time communication, cloud services, and large-scale data sharing. Traditional protocols like TCP (Transmission Control Protocol) paired with TLS (Transport Layer Security) have long served as the backbone for reliable and encrypted data transmission. However, emerging technologies such as QUIC (Quick UDP Internet Connections) promise to address long standing limitations of TCP, such as latency during connection setup and head-of-line blocking, while integrating modern security features like mandatory encryption.

1.1 Motivation

While TCP + TLS remains widely adopted, its performance bottlenecks in high-latency or unstable networks have spurred interest in alternatives. QUIC, built on top of UDP, offers features like 0-RTT connection resumption, multiplexed streams, and built-in encryption with TLS 1.3, positioning it as a compelling candidate for modern applications. However, its trade-offs in terms of resource usage, compatibility, and security robustness are less understood particularly in the context of file transfer systems. This work seeks to evaluate whether QUIC's theoretical advantages translate into practical benefits for file transfer applications compared to the well-established TCP+TLS stack.

2 Background

Secure communication over the Internet has long relied on the interplay between protocols that guarantee reliability and those that ensure privacy. Historically, this has been achieved through the layered combination of TCP (Transmission Control Protocol) and TLS (Transport Layer Security) - a partnership that prioritizes backward compatibility and incremental evolution. In contrast, QUIC (Quick UDP Internet Connections) represents a paradigm shift, integrating transport and security into a unified protocol designed for the demands of modern networks. Understanding their architectural philosophies is key to evaluating their performance and security trade-offs.

2.1 The Layered Approach: TCP + TLS

TCP, the bedrock of Internet communication since the 1970s, was designed to solve a fundamental problem: how to reliably transmit data across inherently unreliable networks [1]. By establishing connections through its three-way handshake, enforcing in-order packet delivery, and dynamically adjusting transmission rates via congestion control algorithms like Cubic and BBR [2, 3], TCP ensures that data arrives intact and sequenced. However, this reliability comes at a cost. For instance, TCP's strict in-order delivery creates head-of-line blocking, where a single lost packet stalls all subsequent data, a bottleneck even more relevant in high-latency or lossy environments like mobile networks.

To address TCP's lack of inherent security, TLS emerged as a cryptographic layer operating on top of it. TLS 1.3, the latest iteration, streamlines the handshake process to a single round trip (or zero for resumed connections) and employs modern encryption suites like AES-GCM and ChaCha20-Poly1305. Yet, because TLS operates as a separate layer, establishing a secure connection requires sequential handshakes: first TCP negotiates the transport channel, then TLS authenticates and encrypts it. For a new connection, this results in 2.5 round trips of latency before any application data is transmitted - a delay that effects in global-scale systems where round-trip times often exceed 100 milliseconds.

This layered design, while modular and backward-compatible, introduces inefficiencies. Middle entities like firewalls and NATs [4], optimized for decades of TCP traffic, often misinterpret or hinder TLS-specific optimizations. Moreover, TCP's head-of-line blocking persists even after TLS encrypts the data, as packet loss at the transport layer stalls the entire encrypted stream. These limitations motivated the search for a protocol that could reimagine both transport and security as a whole.

2.2 QUIC

QUIC, first deployed by Google in 2012 and later standardized as the foundation of HTTP/3, challenges the layered approach of TCP + TLS. Built on top of UDP, a protocol without built-in reliability or congestion control, QUIC integrates transport and cryptographic handshakes into a single layer. This

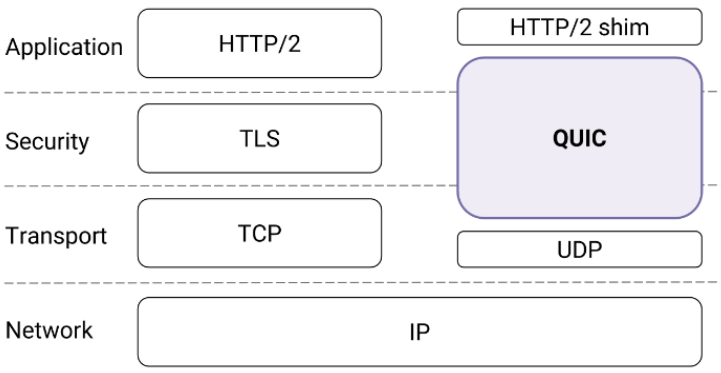

Figure 1: QUIC in the traditional HTTPS stack [4].

integration reflects a design philosophy where security is mandatory, latency is minimized, and reliability is redefined for modern use cases.

From its inception, QUIC mandates encryption. Unlike TCP + TLS, where encryption is an optional add-on, QUIC embeds TLS 1.3 directly into its handshake, ensuring that every connection is secured by default. This allows QUIC to merge the transport and cryptographic handshake into 1 round trip for new connections and 0-RTT for resumed sessions. For example, a user resuming a connection on a mobile network could begin transmitting data immediately, avoiding the 200+ milliseconds of handshake latency typical of the traditional stack. QUIC further diverges from TCP through its handling of multiplexing. By allowing multiple independent streams within a single connection, QUIC eliminates head-of-line blocking at the application layer. If a packet is lost on one stream, other streams continue unaffected - a critical advantage for real-time applications. Additionally, QUIC supports connection migration, enabling seamless transitions between networks without renegotiating security parameters, a feature TCP struggles with due to its strict linkage of connection to IP addresses and ports.

3 Methodology

This section details the implementation of the TCP + TLS and QUIC file transfer clients and servers. Both systems were developed in Python, with careful attention to fairness in testing conditions (e.g., identical file sizes, network emulation settings).

3.1 Workflow and Functional Dynamics

The TCP + TLS implementation relies on synchronous, sequential operations. Clients initiate connections through Python's socket and ssl libraries, wrapping a TCP socket with TLS encryption after the initial handshake. File transfers are structured around JSON-formatted commands (e.g., "action" : "send_file"), with data split into 4 KB chunks for transmission. This approach ensures simplicity but struggles with concurrency, as each connection handles only one stream at a time.

QUIC, powered by the aioquic library, embraces asynchronism. Clients and servers communicate over multiplexed streams within a single QUIC connection, enabling simultaneous uploads and downloads without head-of-line blocking. Commands are sent as plaintext (e.g., upload filename), reducing protocol overhead. The asynchronous model, managed via Python's asyncio, allows QUIC to handle multiple streams concurrently, making it inherently scalable for high-throughput scenarios.

3.2 Security Foundations

Both systems use self-signed RSA certificates, generated automatically, to authenticate servers and establish TLS 1.3 encryption. However, their security workflow differ subtly. In TCP + TLS, the client skips hostname verification for simplicity, accepting any server certificate - a pragmatic choice for testing, but not suitable for production. QUIC enforces certificate validation by default, requiring clients to trust the server's certificate explicitly. This reflects QUIC's design philosophy, where security is mandatory rather than optional.

Encryption parameters are aligned for fairness: AES-GCM secures data in transit, and elliptic-curve Diffie-Hellman ensures forward secrecy. Despite these similarities, QUIC's integration of TLS 1.3 into its handshake eliminates vulnerabilities associated with middleboxes, such as firewalls that mishandle TCP/TLS extensions.

References

[1] Jon Postel. *Rfc0793: Transmission control protocol*. 1981.

[2] Mark Allman, Vern Paxson, and Ethan Blanton. *TCP congestion control*. Tech. rep. 2009.

[3] Neal Cardwell et al. "Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time". In: *Queue* 14.5 (2016), pp. 20–53.

[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*. 2017, pp. 183–196.