Implementation of QUIC Protocol

Software Requirement Specification Document
by Group 16
September 25, 2024

Anvit Gupta
22114009
anvit_g@cs.iitr.ac.in

Ayush Ranjan
22114018
ayush_r@cs.iitr.ac.in

Indranil Das
22114037
indranil_d@cs.iitr.ac.in

Sarvasva Gupta
22114086
sarvasva_g@cs.iitr.ac.in

Souvik Karmakar
22114096
souvik_k@cs.iitr.ac.in

Vineet Kumar
22114107
vineet_k@cs.iitr.ac.in

Abstract

The project focuses on the implementation of the QUIC (Quick UDP Internet Connections) protocol, which is designed to enhance web performance, security, and reliability. As an emerging transport layer protocol, QUIC aims to reduce latency and improve connection efficiency, addressing limitations inherent in traditional protocols such as TCP. Through a systematic approach, we aim to create a robust QUIC implementation that effectively integrates with existing web infrastructures while providing significant improvements in data transmission and connection management. Rigorous testing methodologies will be employed to evaluate performance under various network conditions, ensuring that the implementation meets the demands of modern applications. The project will provide insights into the workings of QUIC, contributing to a deeper understanding of modern network protocols and their implications for future web communication. Ultimately, this implementation serves not only as an academic endeavour but also as a practical tool for enhancing internet connectivity and user experience in real-world applications.

Functional Requirements

1. Connection Management Unit

a. Connection Initialization

- **Creation of QUIC Socket:** Establishes a QUIC socket to manage connections.
- **Listener Object for Server's End:** Listens for incoming connection requests from clients.
- **Requesting and Accepting Connection:** Handles the initiation and acceptance of connections.
- **Cryptographic and Transport Handshake:** Performs the necessary cryptographic operations to securely establish the connection and negotiate transport parameters.

b. Connection Termination

- **Idle Timeout:** Ends connections that are idle for too long.
- **Immediate Close:** Closes the connection immediately when requested.
- **Stateless Reset:** A mechanism to abruptly terminate a connection when the server or client is not able to process the termination request normally.

c. Connection Migration

- **Address and Path Validation:** Manages scenarios where a client's IP address changes (e.g., due to NAT rebinding or network change). Ensures that the new address is valid and the connection can continue securely without interruptions.

2. Stream Management Unit

a. Stream Creation and Destruction

- **Creation:** Sets up data streams within a QUIC connection, allowing data to be sent and received in an ordered and reliable manner.
- **Destruction:** Cleans up and closes streams when they are no longer needed.

b. Sending and Receiving Data

- **Sending Data:** Provides functionality for sending data to the peer.
- **Receiving Data:** Allows the recipient to retrieve data sent by the peer.

c. Flow and Error Control

- **Flow Control:** Manages the rate at which data is sent to prevent overwhelming the recipient.
- **Error Control:** Handles errors that occur during data transmission to ensure reliable communication.

3. Encryption Unit

a. Handling Packet and Header Protection

- **TLS Handshake:** Performs the TLS handshake to establish encrypted communication.
- **TLS Alert:** Manages alerts related to TLS, such as warnings or errors.
- **QUIC Packet Protection:** Encrypts and decrypts QUIC packets to ensure the security of data in transit.

Non-Functional Requirements

1. **Performance** - The implementation must achieve low latency and high throughput to facilitate rapid data exchange between clients and servers. This involves optimizing the transport layer's operations, such as connection establishment and data retransmission. Efficient use of multiplexing should be employed to manage multiple streams concurrently.
2. **Scalability** - The system will be designed to accommodate an increasing number of simultaneous connections without degrading performance. This means it will effectively manage resources such as memory and CPU load as user demand grows. Load testing will be conducted to identify the limits of the implementation and ensure it can handle bursts of traffic, allowing for a smooth user experience even during peak usage times.

3. **Reliability** - Reliability is critical for ensuring that data is transmitted accurately and consistently. The implementation will incorporate mechanisms for detecting and recovering from packet loss. It will also maintain state information to allow for seamless reconnection in the event of network disruptions.

4. **Security** - Given the importance of secure data transmission, the implementation must incorporate strong encryption protocols to protect data in transit. We will implement integrity checks to verify that data has not been tampered with during transmission. Additionally, measures against common threats such as Distributed Denial of Service (DDoS) attacks and man-in-the-middle attacks will be in place, ensuring that the system is resilient against malicious activities.

5. **Interoperability** - The implementation will be compatible with existing QUIC standards and other related protocols, such as HTTP/3. This ensures that it can communicate effectively with other QUIC-based systems and clients, promoting a seamless experience for end users. Compliance with established IETF specifications will help guarantee that the implementation can work in diverse environments and with various client libraries.

6. **Usability** - The system will be designed with user experience in mind, offering an intuitive configuration process and clear documentation. This includes well-structured guides and examples that demonstrate how to set up and troubleshoot the QUIC implementation.

7. **Maintainability** - To facilitate long-term support and updates, the codebase will be organized, modular, and adhere to best coding practices. Comprehensive documentation will be provided, detailing the architecture, functions, and expected behaviors of different components. A clear version control strategy will be established to manage changes and ensure that new features can be added without disrupting existing functionality.

8. **Resource Efficiency** - The implementation must use system resources judiciously, optimizing CPU and memory usage while maintaining high performance. This includes minimizing overhead associated with data processing and connection management. Profiling and benchmarking will be performed to identify bottlenecks and areas where resource usage can be improved, particularly on devices with limited capabilities.

9. **Latency** - The system aims for low round-trip times (RTT) to enhance user experience during data exchanges. This involves optimizing the connection handshake process to reduce setup times and employing strategies to minimize delays during data transmission.

10. **Compliance** - The implementation will adhere to relevant industry standards and protocols, particularly those set forth by the Internet Engineering Task Force (IETF). This includes following the specific requirements outlined in the QUIC and HTTP/3 specifications. Ensuring compliance not only facilitates interoperability with other systems but also promotes a higher level of trust and reliability in the implementation.