Implementation of QUIC Protocol

Design 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

1. Summary of Design and Architecture of QUIC

- Our project focuses on implementing the QUIC protocol, which aims to provide reliable, low-latency communication over UDP. QUIC, which stands for Quick UDP Internet Connections, integrates features traditionally handled by TCP and TLS, providing significant performance benefits, especially in scenarios involving web traffic.

- The design architecture consists of three main components: a QUIC client, a QUIC server, and the underlying UDP transport layer. The client initiates connections, handles application data, and manages connection states, while the server responds to client requests and maintains session integrity. We utilize UDP for transport due to its lightweight nature, enabling faster packet transmission.

- Each component communicates using QUIC frames, which encapsulate various types of data such as stream data, control messages, and connection management information.

- Key features include connection multiplexing, which allows multiple streams within a single connection, and 0-RTT connection establishment for faster reconnections. Security is ensured with built-in TLS encryption ensuring all data is protected during transmission.

2. High Level Design

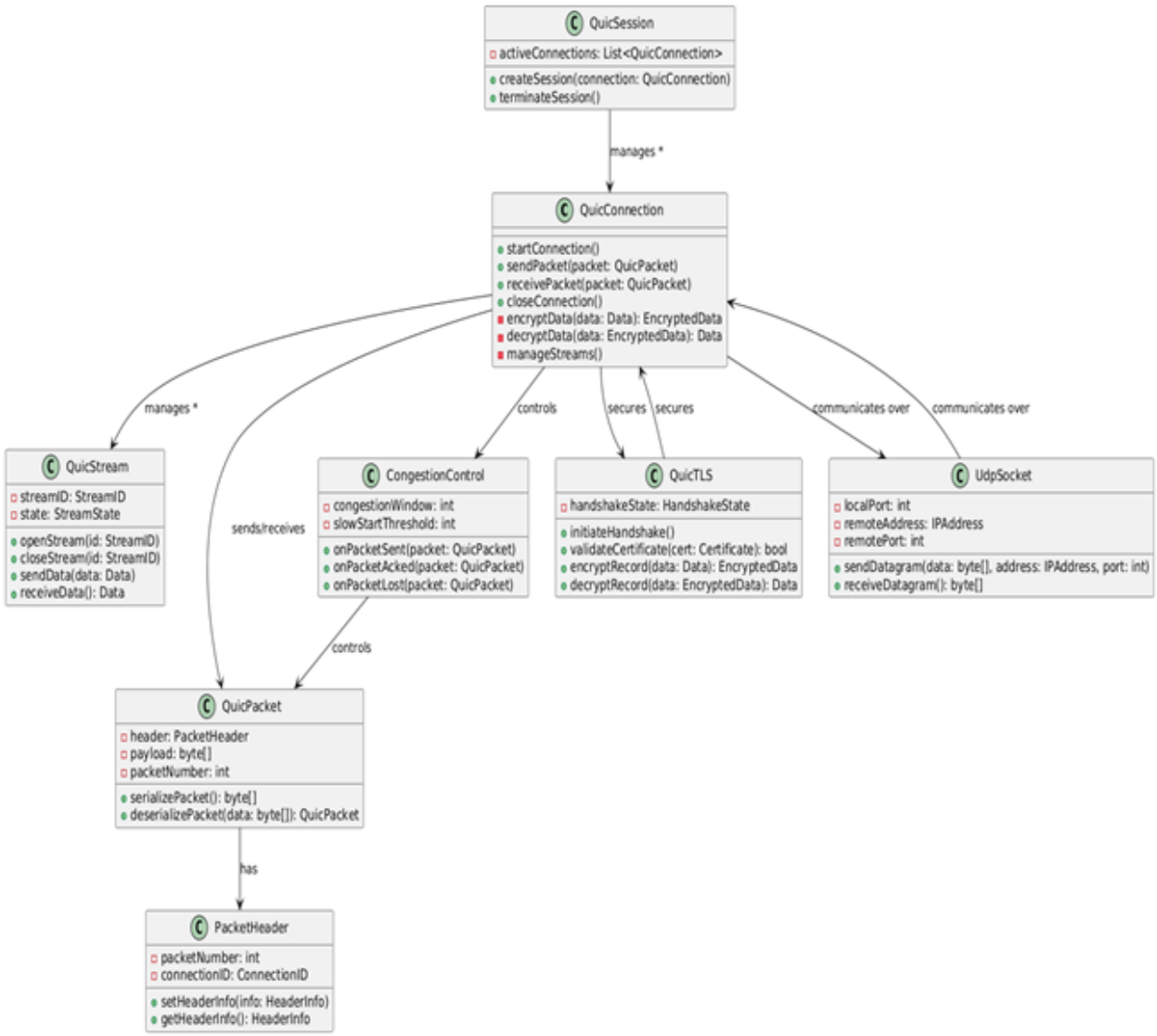

Figure 1: Collaboration Diagram

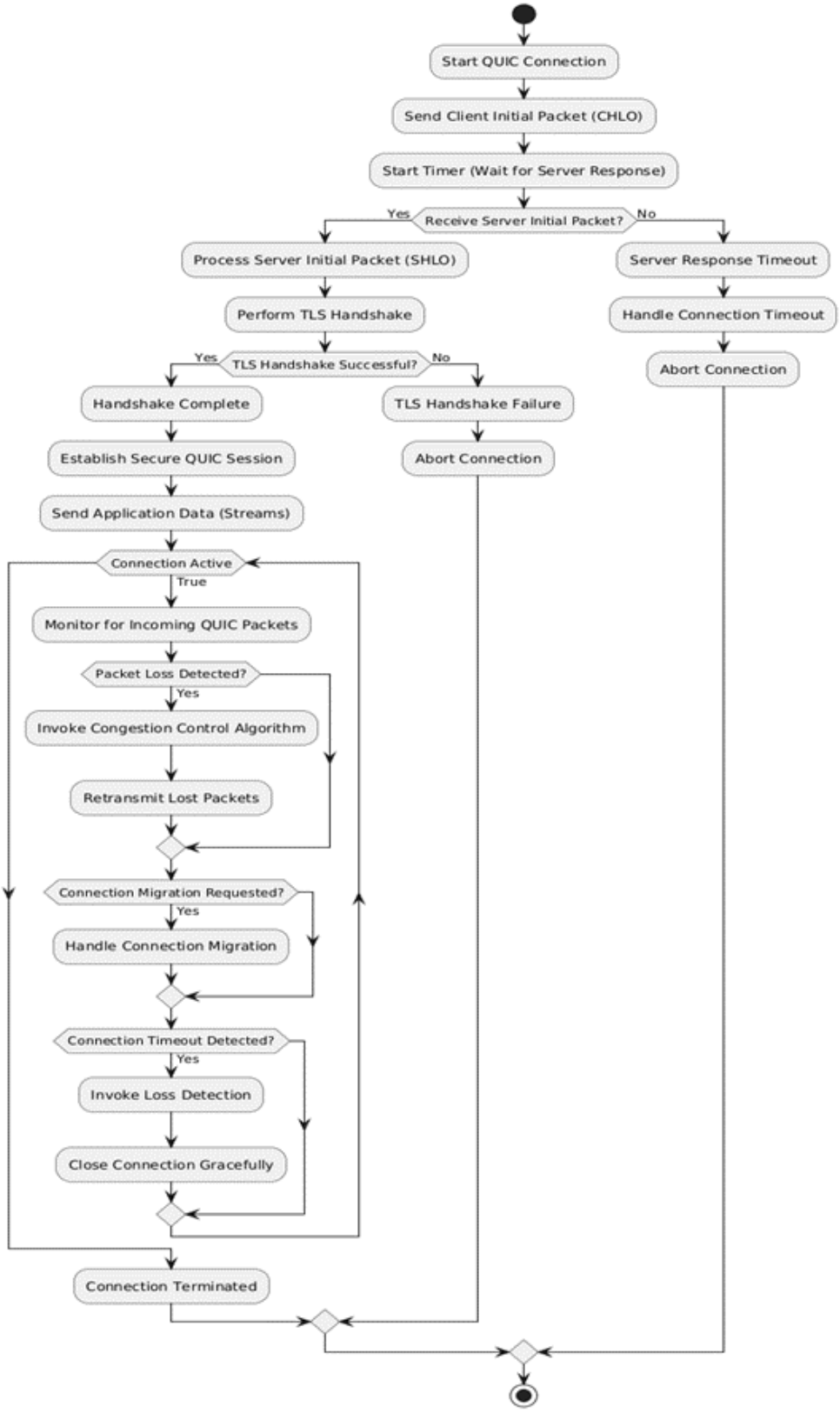

Figure 2: Flow Chart

3. Low Level Design

1. Packet Structure

- **Objective:** Define how QUIC packets will be structured and processed.
- **Components:**
  - **Header Structure:**
    Packet Number
    Connection ID (64-bit)
    Flags (indicating version, type, etc.)
  - **Payload:**
    Encrypted payload using AEAD
    Stream frames (data frames)
    Acknowledgment (ACK) frames
  - **Retry and Token Frames:**
    For address validation, include Retry and NEW_TOKEN frames.

2. Connection Management

- **Objective:** Manage QUIC connections and their states.
- **Components:**
  - **Connection ID Generation:**
    Implement logic for generating and managing 64-bit connection IDs.
    Handle migration of connection IDs during network changes.
  - **Connection State:**
    Maintain the connection states: Idle, Handshake, Established, Closing, Closed.
    Manage active streams and flow control for each connection.
  - **Multiplexing:**
    Support multiple streams over a single connection using stream IDs and priority management.

3. Handshake & Encryption

- **Objective:** Handle secure handshake using TLS 1.3 and ensure encrypted communication.
- **Components:**

- **TLS 1.3 Integration:**
  Use TLS 1.3 for the handshake, integrating with the QUIC transport layer.
  Store keys (0-RTT, 1-RTT) in QUIC and TLS context.

- **CRYPTO Frames:**
  Exchange TLS handshake messages via CRYPTO frames during the initial handshake.

- **Key Management:**
  Support key updates and key discard mechanisms for 0-RTT, Handshake, and 1-RTT keys.

4. Packet Sending and Receiving

- **Objective:** Handle the processing and transmission of QUIC packets.
- **Components:**
  - **Sending Logic:**
    Fragment outgoing data into QUIC STREAM frames.
    Apply packet protection using AEAD encryption.
    Manage retransmission of lost packets based on acknowledgments.

  - **Receiving Logic:**
    Parse incoming QUIC packets.
    Reassemble STREAM frames from incoming data.
    Handle out-of-order packet receipt and retransmission using ACK frames.
    Process connection migration when packets arrive from a new IP address.

  - **Congestion Control:**
    Use a congestion control algorithm (e.g., TCP Cubic).
    Implement pluggable congestion control modules for future extension.

5. Flow Control

- **Objective:** Implement flow control to manage data transmission for both streams and connections.
- **Components:**
  - **Stream-Level Flow Control:**
    Use stream-specific flow control windows.
    Dynamically adjust window sizes based on the receiver's buffer space.

- **Connection-Level Flow Control:**
  Apply overall limits to avoid overloading the connection with too much data at once.

- **Credit Updates:**
  Send and receive WINDOW_UPDATE frames to manage flow control credits.

6. Loss Recovery

- **Objective:** Detect and recover from packet losses.
- **Components:**
  - **ACK Frames:**
    Maintain a list of sent packets and detect losses using ACK frames.

  - **NACK Ranges:**
    Handle up to 256 NACK ranges to detect reordering or loss of packets.

  - **Retransmission:**
    Retransmit lost packets, differentiating between original and retransmitted packets using sequence numbers.

  - **Forward Error Correction (Optional):**
    Support FEC to recover from packet loss without waiting for retransmission, using parity packets.

7. Connection Migration

- **Objective:** Handle connection migration when the client's IP address changes.
- **Components:**
  - **Connection ID:**
    Retain connection ID across IP address changes to continue communication.

  - **Address Validation:**
    Implement logic for address validation using tokens during connection migration.

8. Security and Error Handling

- **Objective:** Secure communication and handle errors gracefully.
- **Components:**

- **AEAD Encryption:**
  Encrypt packets using AEAD algorithms negotiated via TLS.

- **Integrity Checks:**
  Ensure packet headers and payloads are protected using separate keys.

- **Connection Termination:**
  Handle QUIC CONNECTION_CLOSE frames for both graceful and abrupt connection closures.

- **Error Codes:**
  Implement error handling with appropriate error codes for various connection states and events.

9. Testing and Debugging Utilities

- **Objective:** Provide tools for debugging and performance testing.
- **Components:**
  - **Packet Tracing:**
    Implement a module to log QUIC packet headers and payloads for debugging.

  - **Performance Metrics:**
    Measure connection establishment time, packet loss rates, retransmissions, etc.

  - **Error Logging:**
    Maintain detailed logs for connection errors and state transitions.