# Implementation of QUIC

## Final Report

## Group 16

## Team members' Details:


1. **Anvit Gupta, 22114009**
   Mobile No - +91 94620 11044
   Email ID – anvit_g@cs.iitr.ac.in
   Contribution – Worked on Connection Management Feature and System Testing.

2. **Vineet Kumar, 22114107**
   Mobile No - +91 92634 36403
   Email ID – Vineet_k@cs.iitr.ac.in
   Contribution - Worked on System Design and Protocol level Programs and Thread Functions. Built the Connection Management Feature

3. **Ayush Ranjan, 22114018**
   Mobile No - +91 74829 58551
   Email ID – ayush_r@cs.iitr.ac.in
   Contribution – Worked on client and server modules along with Documentation.

4. **Souvik Karmakar, 22114096**
   Mobile No - +91 86429 24659
   Email ID – souvik_k@cs.iitr.ac.in
   Contribution – Worked on System Design and stream management and flow control

5. **Sarvasva Gupta, 22114086**
   Mobile No - +91 79899 04811
   Email ID – sarvasva_g@cs.iitr.ac.in
   Contribution – Worked on Packets and Frame Handling Unit. Helped with Initial Design

6. **Indranil Das, 22114037**
   Mobile No - +91 89020 31812
   Email ID – indranil_d@cs.iitr.ac.in
   Contribution – Helped with Testing and Debugging.

# Introduction

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 endeavor but also as a practical tool for enhancing internet connectivity and user experience in real-world applications

# Overview of QUIC

QUIC (Quick UDP Internet Connections) is a modern transport protocol first proposed by Google in 2012 and in 2021, the IETF officially published QUIC as RFC 9000. QUIC was motivated in large part by the challenges of matching the request/response semantics of HTTP to the stream-oriented nature of TCP. These issues have become more noticeable over time, due to factors such as the rise of high-latency wireless networks, the availability of multiple networks for a single device (e.g., Wi-Fi and cellular), and the increasing use of encrypted, authenticated connections on the Web.

QUIC implements fast connection establishment with encryption and authentication in the first RTT. It provides a connection identifier that persists across changes in the underlying network. It supports the multiplexing of several streams onto a single transport connection, to avoid the head-of-line blocking that may arise when a single packet is dropped while other useful data continues to arrive. And it preserves (and in some ways improves on) the congestion avoidance properties of TCP.

# Motivation for the Project

The motivation for implementing the QUIC protocol stems primarily from its ability to significantly improve performance, reduce latency, and enhance security.

QUIC is the most interesting development in the world of transport protocols. Many of the limitations of TCP have been known for decades, but QUIC represents one of the most successful

efforts to date to stake out a different point in the design space. Because QUIC was inspired by experience with HTTP and the Web—which arose long after TCP was well established on the Internet—it presents a fascinating case study in the unforeseen consequences of layered designs and in the evolution of the Internet.

QUIC offers faster connection setup with reduced round trips, minimizes latency through multiplexed streams, and eliminates head-of-line blocking, which is common in TCP-based protocols like HTTP/2. By utilizing UDP, QUIC allows for more efficient handling of packet loss and congestion control, providing better performance in high-latency or lossy networks. It also integrates encryption as a core feature, ensuring strong security with TLS 1.3 and forward secrecy by default. Moreover, QUIC supports connection migration, making it ideal for mobile or dynamic network environments where users frequently switch between networks without interrupting active sessions. With its foundation for HTTP/3, QUIC is positioned as a future-proof protocol for modern web applications, offering better scalability, lower overhead, and enhanced user experience, particularly for high-traffic or real-time services.

## Application Domain

The **QUIC (Quick UDP Internet Connections)** protocol, originally developed by Google and later standardized by the IETF, offers a range of benefits over traditional TCP-based protocols, particularly in terms of low latency, better multiplexing, and improved security. QUIC has a wide range of potential application domains.

**1. Web Browsing and Content Delivery**

- **HTTP/3**: QUIC is the foundation of HTTP/3, the latest version of the HTTP protocol. By using QUIC, HTTP/3 significantly reduces connection establishment times and improves load speeds, especially over mobile networks or high-latency connections.
- **Content Delivery Networks (CDNs)**: QUIC can be used by CDNs to accelerate the delivery of web content, media, and files, providing faster, more reliable access to global audiences.
- **Web Optimization**: QUIC's multiplexing and congestion control mechanisms improve the performance of modern web applications, which often rely on numerous parallel connections for various resources.

**2. Video Streaming and Real-time Media**

- **Video Streaming Services**: Services like YouTube, Netflix, and other media platforms can benefit from QUIC's reduced connection setup time and more efficient handling of video streams, particularly for mobile devices.
- **Live Streaming**: QUIC is useful for low-latency applications like live streaming (e.g., gaming, sports events, or real-time broadcasting) where reducing buffering time and improving stream stability are critical.
- **WebRTC**: QUIC can also be used to improve WebRTC (Web Real-Time Communication)

applications, ensuring lower latency and more stable peer-to-peer connections for video conferencing, online collaboration, and VoIP services.

**3. Mobile Networking**

- **Mobile Applications**: QUIC's lower latency and reduced connection overhead make it ideal for mobile environments where connection setup times, packet loss, and fluctuating network conditions are challenges.
- **Network Optimizations for 4G/5G**: QUIC is particularly well-suited for the high-speed, low-latency requirements of 5G networks, offering an effective solution for optimizing applications that need to make the most of such networks.

**4. Cloud Services and Microservices**

- **Cloud Application Communication**: QUIC can be used for high-performance communication between cloud services or microservices, ensuring fast and secure data transfer, especially in environments where multiple services are interacting concurrently.
- **Edge Computing**: In edge computing architectures, where low-latency communication is paramount, QUIC can be used to enhance communication between edge nodes and central servers, ensuring timely processing and data transmission.

**5. IoT and Embedded Systems**

- **Low-Latency Communication for IoT Devices**: QUIC's low latency and stream multiplexing make it a good fit for Internet of Things (IoT) applications that require fast, reliable communication between devices, especially in scenarios with high numbers of devices and intermittent network connectivity.
- **Security in IoT**: QUIC natively integrates encryption, which is beneficial for securing IoT communications, where security can be a significant concern due to the limited processing power of many devices.

**6. Gaming and Real-Time Applications**

- **Online Multiplayer Games**: QUIC's reduced latency and more efficient handling of packet loss make it an excellent choice for online gaming applications, which require real-time communication with minimal delay.
- **Augmented and Virtual Reality (AR/VR)**: QUIC is well-suited for AR/VR applications, where low-latency communication is essential for delivering immersive experiences with minimal lag or interruptions.

**7. VPNs and Secure Networking**

- **VPNs (Virtual Private Networks)**: QUIC can be used in VPN implementations to improve speed and reliability over traditional VPN protocols (like OpenVPN or IPsec), particularly when dealing with high-latency networks.
- **Secure Data Transmission**: QUIC is designed with encryption built into the protocol,

making it ideal for secure communication over the internet, which is essential for protecting sensitive data in transit.

**8. Cloud Storage and File Transfer**

- **File Transfers**: QUIC's multiplexing feature can improve the performance of file transfer protocols by allowing multiple streams to be sent over a single connection, leading to faster transfer speeds and better handling of packet loss.
- **Backup and Sync Services**: QUIC could be adopted in backup and synchronization applications, where low-latency and efficient handling of multiple concurrent file transfers are crucial.

**9. Content Security and Privacy**

- **DDoS Mitigation**: QUIC's connection multiplexing and encryption features can help mitigate Distributed Denial of Service (DDoS) attacks and offer additional security features like resistance to some types of traffic analysis and eavesdropping.
- **Privacy-Enhanced Browsing**: QUIC's built-in encryption can provide additional privacy for users, which could be important for privacy-focused applications and services, such as secure messaging or privacy-oriented search engines.

**10. Corporate Networks**

- **Internal Corporate Communication**: QUIC can be adopted in internal corporate applications to improve the performance and security of communications, especially in organizations with distributed teams or remote workers.
- **Remote Desktop and Virtualization**: Protocols based on QUIC can be used for virtual desktop infrastructures (VDIs), remote desktop solutions, or virtual machine environments, enabling fast, efficient access to resources in corporate networks.

**11. Edge and CDN Integration**

- **Distributed CDN Systems**: QUIC is well-suited for use in edge-based CDNs, where minimizing latency between edge nodes and users is critical. The protocol's features can help ensure better user experiences in real-time applications.
- **Optimized Caching**: QUIC can help improve caching strategies by enabling better resource allocation, faster access to cached content, and reduced load times for end-users.

**12. DNS Over QUIC**

- **Secure DNS Resolution**: QUIC is also used in DNS over QUIC, which aims to provide a more secure and private alternative to traditional DNS over UDP or TCP, reducing the chances of DNS interception or man-in-the-middle attacks.

# Design of the System

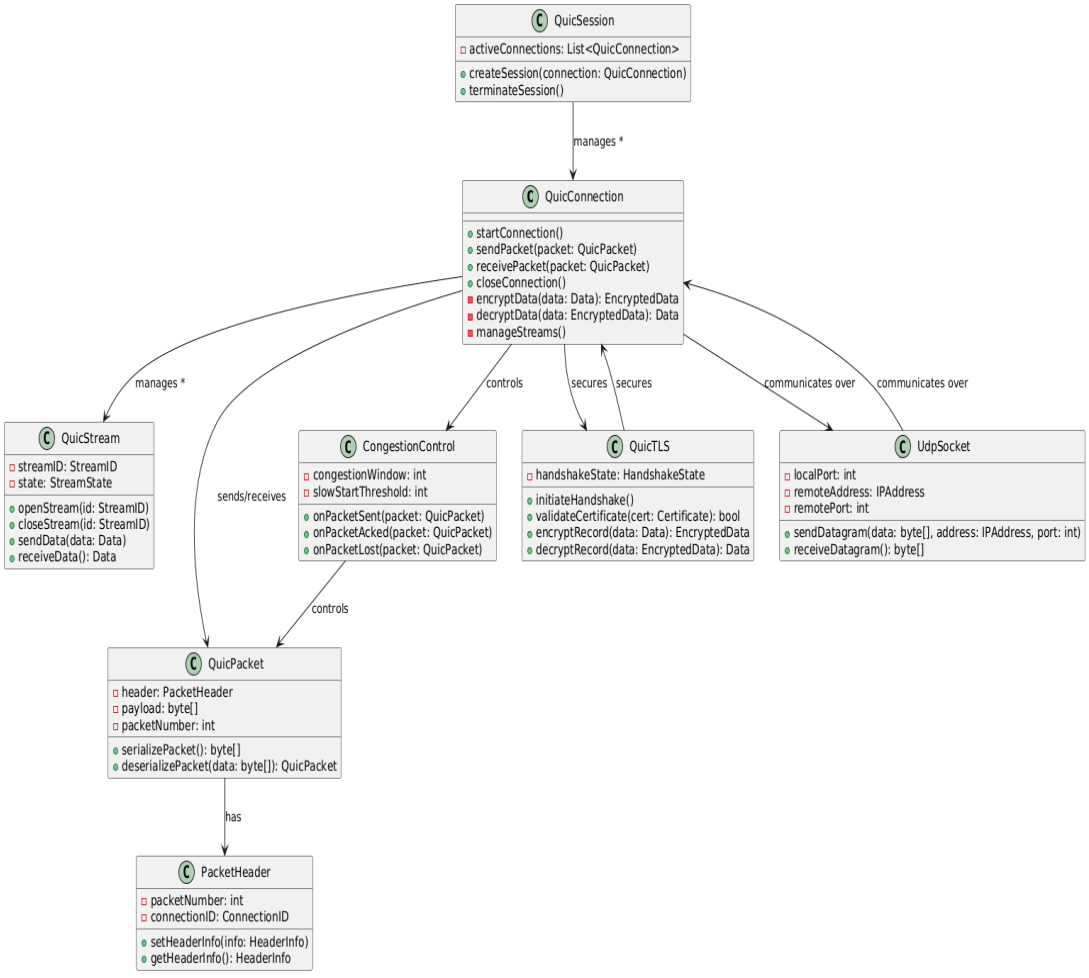
The design architecture consists of three main components: a QUIC client, a QUIC server, and the underlying UDP transport layer. The client initiates connections, handles application data, and manages connection states, while the server responds to client requests and maintains session integrity. We 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.

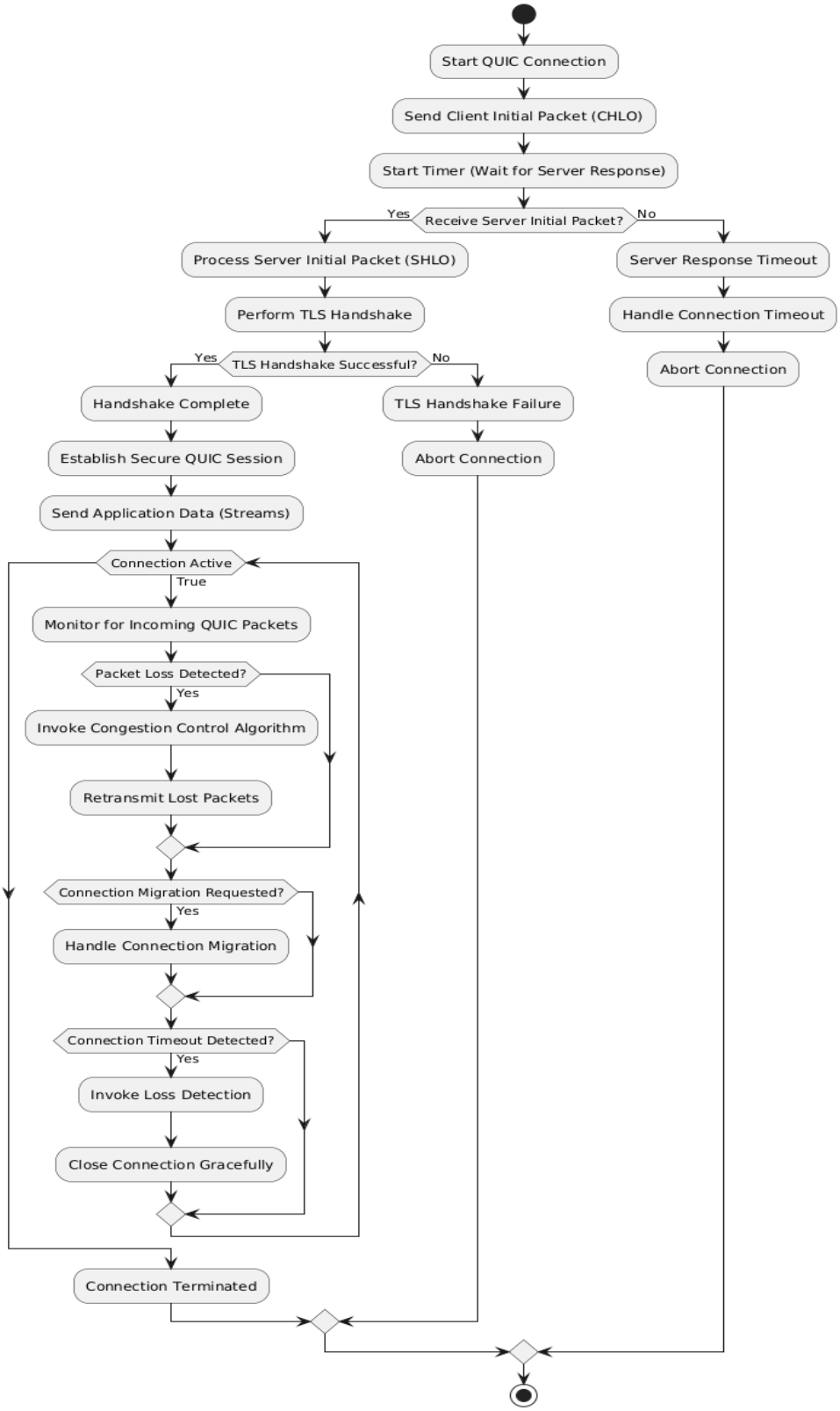
## High-level design

Security is ensured with built-in TLS encryption ensuring all data is protected during transmission.

- Class diagram -



- Control Flow diagram



# 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.

# Tools and Technologies Used in the System

This project utilized a variety of tools and technologies, ensuring a robust, efficient, and scalable development process. These tools facilitated protocol design, testing, performance optimization, and security verification.

**1. Programming Languages**

- **C/C++**: For the core implementation of the QUIC protocol, we used C/C++ due to their performance efficiency and fine-grained control over memory and system resources, which are crucial for optimizing network protocols.
- **Python**: Python was used for writing automated testing scripts and developing performance monitoring tools. Its rich ecosystem of libraries allowed for quick development and integration.

**2. Cryptographic and Security Libraries**

- **OpenSSL**: OpenSSL was employed to handle encryption and cryptographic operations required by QUIC's built-in TLS. It ensured secure data transmission with up-to-date cryptographic standards and protocols.
- **Wireshark**: For packet analysis and debugging, Wireshark was invaluable. It allowed us to monitor and inspect encrypted QUIC traffic to verify handshake processes, packet retransmissions, and overall protocol behaviour.

**3. Development and Testing Tools**

- **Visual Studio Code**: VS Code served as our primary integrated development environment (IDE), chosen for its versatility and support for multiple languages and extensions that improved productivity.
- **Git**: For version control, Git allowed efficient team collaboration, version tracking, and branching, facilitating a seamless workflow and code management throughout the development process.

# Conclusion

The successful implementation of the QUIC protocol in our project has led to substantial advancements in network efficiency, security, and adaptability. By leveraging QUIC's 0-RTT and 1-RTT handshake mechanisms, we were able to reduce connection setup time considerably, resulting in lower latency and a more responsive experience for real-time applications like video streaming and online gaming. Additionally, QUIC's stream multiplexing feature eliminated head-of-line blocking, allowing faster and more efficient resource loading, particularly beneficial for web applications.

Another significant achievement was our successful implementation of connection migration,

which allows for uninterrupted data transfers even when a device switches networks. This feature was crucial for mobile use cases, ensuring seamless connectivity and continuous performance for applications that require persistent communication, like video calls or IoT devices. Overall, our project lays a solid foundation for future research and practical deployments, demonstrating QUIC's potential to reshape modern network protocols and improve the user experience across a variety of applications.

# Bibliography


a. RFC8999 - https://datatracker.ietf.org/doc/rfc8999/
b. RFC9000 - https://datatracker.ietf.org/doc/rfc9000/
c. RFC9001 - https://datatracker.ietf.org/doc/rfc9001/
d. RFC9002 - https://datatracker.ietf.org/doc/rfc9002/
e. Linux Man Pages - https://man7.org/linux/man-pages/man2/