# Extra Reading- Week 3

## 1. Introduction

* QUIC (Quick UDP Internet Connections) is a transport protocol developed by Google.
* Designed to improve HTTPS traffic performance and support rapid deployment of transport innovations.
* QUIC is an encrypted, multiplexed, and low-latency protocol that replaces TCP, TLS, and HTTP/2.
* It has been globally deployed by Google on thousands of servers, handling traffic from Chrome, YouTube, and Google Search.
* As of the study, 7% of global Internet traffic is QUIC.

## 2. Motivation for QUIC

* Latency Reduction: The web is increasingly latency-sensitive.
* Limitations of TCP/TLS:
* Protocol ossification: Middleboxes make updating TCP difficult.
* Implementation delays: TCP changes require OS upgrades, slowing adoption.
* Handshake delays: TLS and TCP handshakes introduce additional latency.
* Head-of-line blocking: TCP enforces strict packet order, slowing multiplexed connections.
* QUIC's Approach:
* Built on UDP to avoid TCP’s ossification.
* User-space implementation for faster updates.
* 0-RTT handshakes for repeat connections.
* Multiplexed streams to avoid head-of-line blocking.

## 3. QUIC Design and Features

### 3.1 Connection Establishment

* Combines cryptographic and transport handshakes.
* Uses Diffie-Hellman key exchange for encryption.
* 0-RTT handshake allows instant data transmission on repeat connections.

### 3.2 Stream Multiplexing

* Unlike TCP, QUIC supports multiple streams per connection.
* Independent streams prevent one packet loss from blocking others.

### 3.3 Authentication and Encryption

* Fully encrypted and authenticated transport.
* Prevents middlebox interference and protocol ossification.

### 3.4 Loss Recovery

* QUIC improves upon TCP’s retransmission ambiguity.
* Uses unique packet numbers and ACK-based feedback for accurate loss detection.

### 3.5 Flow Control

* Implements stream-level and connection-level flow control to prevent buffer overflow.

### 3.6 Congestion Control

* Uses Cubic congestion control, similar to TCP.
* Can support alternative congestion control algorithms.

### 3.7 NAT Rebinding & Connection Migration

* QUIC supports connection migration across different networks using a Connection ID.

### 3.8 QUIC Discovery for HTTPS

* Websites advertise QUIC support using the Alt-Svc HTTP header.
* Chrome attempts QUIC connections first but falls back to TLS/TCP if blocked.

## 4. Experimentation Framework

* QUIC was A/B tested through Chrome's experimentation framework.
* Metrics analyzed:
* HTTP error rates.
* Transport latency.
* Video rebuffer rates.
* Google’s global server fleet collected real-time performance data.

## 5. Internet-Scale Deployment

* QUIC Deployment Timeline:
* 2013: Experimental launch in Chrome.
* 2014: Limited A/B testing.
* 2017: Used in 30% of Google’s traffic.
* Challenges Encountered:
* December 2015: Bug caused unencrypted 0-RTT requests, forcing Google to disable QUIC.
* Mobile QUIC adoption: YouTube and Google Search apps expanded QUIC usage.

## 6. QUIC Performance Analysis

### 6.1 Search Latency Reduction

* Desktop search latency reduced by 8.0%.
* Mobile search latency reduced by 3.6%.
* Larger benefits in high-latency networks.

### 6.2 Video Playback Improvements

* YouTube startup latency reduced by 8.0% (desktop) and 5.3% (mobile).
* YouTube rebuffer rates improved by 18.0% (desktop) and 15.3% (mobile).
* QUIC’s loss recovery helped maintain smooth video playback.

### 6.3 Regional Performance

* Higher latency networks benefit more from QUIC.
* Example: India saw a 13.2% search latency reduction, while South Korea saw only 1.3%.

### 6.4 Server CPU Utilization

* Initially 3.5× more CPU-intensive than TLS/TCP.
* Optimized with:
* ChaCha20 encryption for efficiency.
* Packet batching to reduce overhead.
* Better memory management.
* After optimizations, QUIC’s CPU cost is now only 2× that of TLS/TCP.

## 7. Lessons Learned

### 7.1 UDP Blockage and Throttling

* 4.4% of networks block QUIC (mainly corporate firewalls).
* 0.3% of ISPs throttle UDP traffic.

### 7.2 Middlebox Ossification

* Some firewalls started blocking QUIC incorrectly, causing unexpected packet loss.

### 7.3 Failed Forward Error Correction (FEC) Experiment

* FEC (error correction) increased CPU usage but did not significantly improve performance.
* QUIC removed FEC support after testing.

### 7.4 Rapid Iteration and Deployment

* User-space QUIC allowed fast bug fixes and updates.
* Version tracking showed QUIC was updated much faster than TCP.

## 8. Future Work

* Further CPU optimizations for QUIC servers.
* Better congestion control algorithms.
* Improving QUIC on mobile networks.
* Path MTU discovery to support larger packet sizes.

## Conclusion

* QUIC successfully improves HTTPS performance, especially in high-latency networks.
* Widely adopted in Google services like Chrome, YouTube, and Google Search.
* Overcame TCP/TLS limitations, but faces challenges like UDP blocking.
* Standardized by IETF to ensure widespread adoption across the Internet.