Spring 2026

COMP6103P
Advanced Computer Networking

Advanced Internet Protocols

From Heuristic to Machine Learning Driven

Topic Outline (~3 Lectures)

- **Background**
- **From TCP to QUIC**
  - The QUIC Transport Protocol: Design and Internet-Scale Deployment, SIGCOMM 2017
- **Learning-based Congestion Control**
  - PCC: Re-architecting Congestion Control for Consistent High Performance, NSDI 15
  - PCC Vivace: Online-Learning Congestion Control, NSDI 17
- **Application-Driven Network Optimization**
  - Neural Adaptive Video Streaming with Pensieve, SIGCOMM 2017

Internet Features

- **Heterogeneous Network Environment**
  - Wired vs Wireless
  - High bandwidth vs Low bandwidth
  - Stable vs Dynamic networks
  - Long RTT vs Short RTT
- Making it **difficult for a single protocol design to perform well in all scenarios**.

Internet Features

- **Dynamic and Unpredictable Conditions**
  - Congestion changes rapidly
  - Bandwidth fluctuates
  - Packet loss patterns vary
- Traditional TCP protocols rely on **static rules**, which are not suitable for **dynamic environments**.

Internet Features

- **Diverse Application Requirements**
  - File downloading: High throughput
  - Game (CS/Dota): Low latency
  - Video streaming: **Quality of User Experience**
- Different applications have different performance objectives, but TCP provides a **one-size-fits-all solution**.

Limitations of TCP

- TCP has been running for over 40 years!
  - Too complicated
  - Not meet the diverse requirements from various applications
  - Not tailored for some important applications

Efforts

- Redesign the transport layer protocol to overcome TCP limitations
  - **QUIC**
- Replace rule-based congestion control with learning-based control
  - **PCC**
  - **PCC Vivace**
- Application learns optimal network behavior
  - **Pensieve**

Reading Materials

- **Extension to QUIC**
  - XLINK: QoE-Driven Multi-Path QUIC Transport in Large-scale Video Services, SIGCOMM 21
- **More Video Streaming Applications:**
  - Optimizing Low-Latency Video Streaming: AI-Assisted Codec-Network Coordination, SIGCOMM 25 Tutorial (complete video is available)
- **More ML-Driven CCs:**
  - Lots of papers from our group
  - Particular papers from Prof. Han Tian: *Spine, Jury, Astraea, PolicyCache*

The QUIC Transport Protocol: Design and Internet-Scale Deployment

Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, Zhongyi Shi

Google, SIGCOMM 2017

Outline

- **Introduction and Motivation**
- Design and Implementation
- Internet-scale Deployment and Performance
- Experience

Introduction

- A user-space, encrypted transport protocol on UDP
- Developed by Google
- Deployed at Google's front-end servers and applications
  - Chrome, Google Search, YouTube app. etc.
  - Constitute 7% Internet traffic

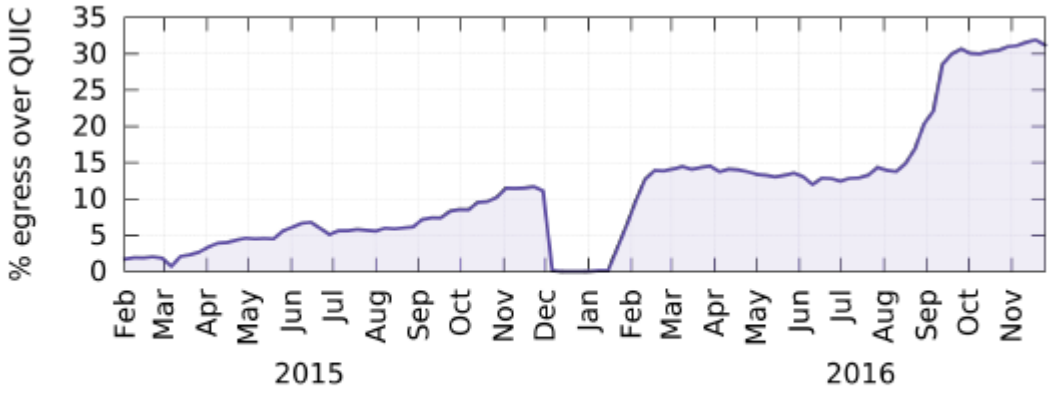


Figure 2: Timeline showing the percentage of Google traffic served over QUIC. Significant increases and decreases are described in Section 5.1.

Introduction

- Google's design
  - QUIC as a whole
  - Replace most of the traditional HTTPS stack: HTTP/2, TLS, and TCP
- IETF standardization
  - Modularize it into constituent parts.
  - A series of RFCs.
- Focus on pre-IETF QUIC (Google QUIC)

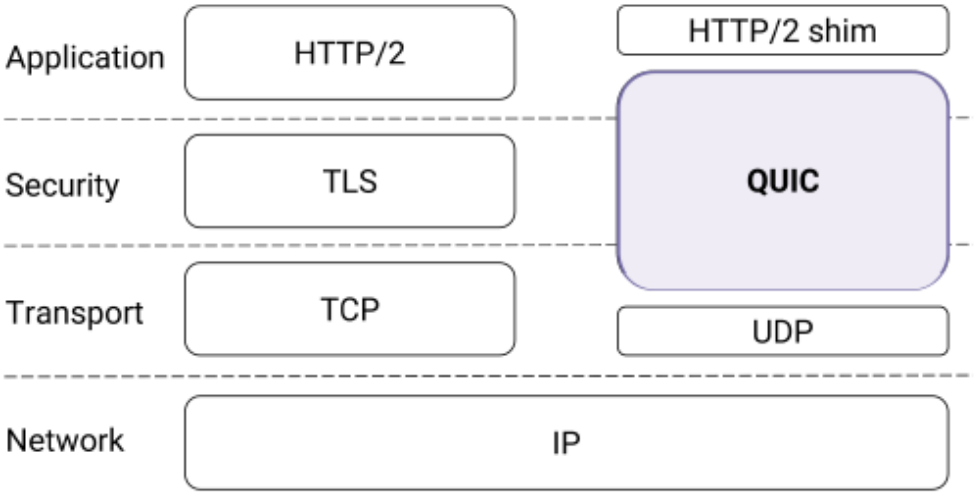


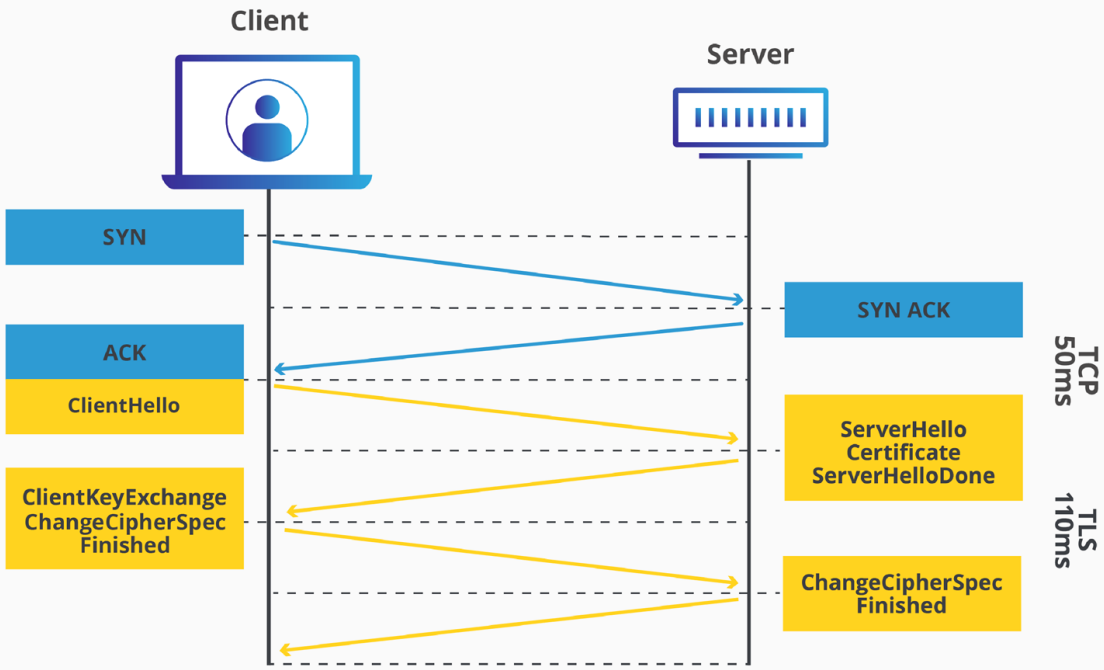
Figure 1: QUIC in the traditional HTTPS stack.

Motivation

- **Protocol Entrenchment**
  - Deploying changes to TCP has reached a point of diminishing returns, where simple protocol changes are now expected to **take upwards of a decade to see significant deployment**
  - Middlebox: firewall, NAT, IDS
- **Implementation Entrenchment**
  - Coupling of the transport implementation to the OS **limits deployment velocity of TCP changes**

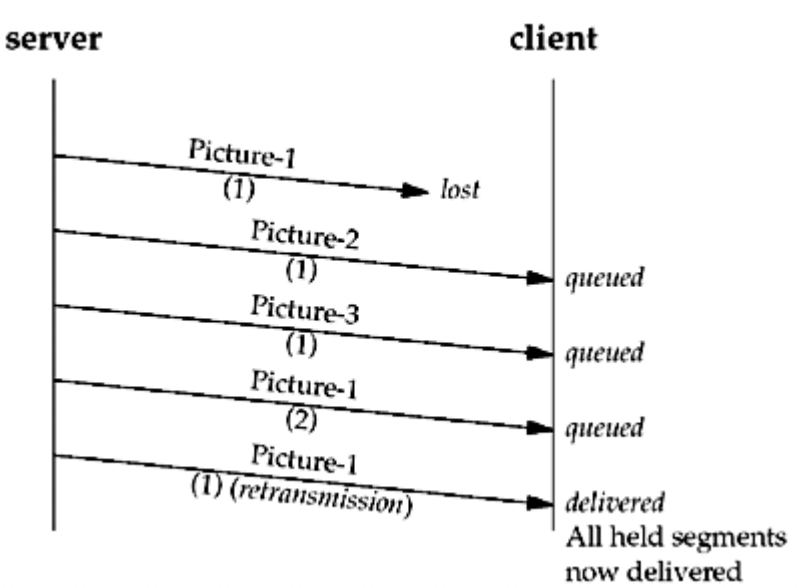
Motivation

- **Handshake Delay**
  - TCP connections commonly incur at least one round-trip delay of connection setup time before any application data can be sent, and TLS adds **two round trips** to this delay



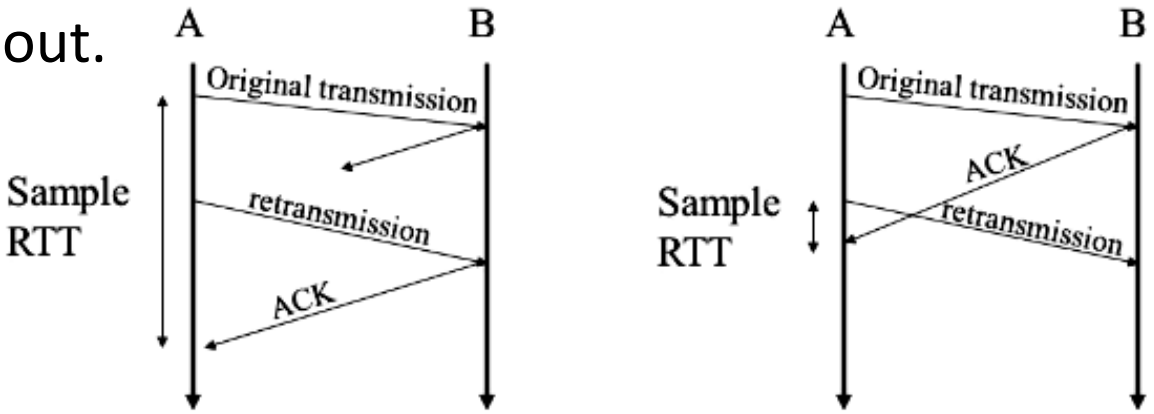
Motivation

- **Head-of-line Blocking Delay** at TCP level
  - HOL problem at the HTTP level solved by pipelining
  - Parallel requested HTTP objects delivery on a single TCP stream
  - TCP stack has to ensure all the bytes have been received in order before delivering to HTTP.
  - An object delivery may be blocked by packet loss of another object delivery.



Motivation

- **Retransmission Ambiguity**
  - A retransmitted TCP segment carries the same sequence numbers as the original packet.
  - The receiver of a TCP ACK cannot determine whether the ACK was sent for the original transmission or for a retransmission
    - Sample RTT for this ACK is ambiguous
  - Loss of a retransmitted segment is commonly detected via an expensive timeout.



Quick Summary

- **TCP (Internet) suffers from the following problems:**
  - Protocol Entrenchment
  - Implementation Entrenchment
  - Handshake Delay
  - Head-of-line Blocking Delay at TCP level
  - Retransmission ambiguity

Outline

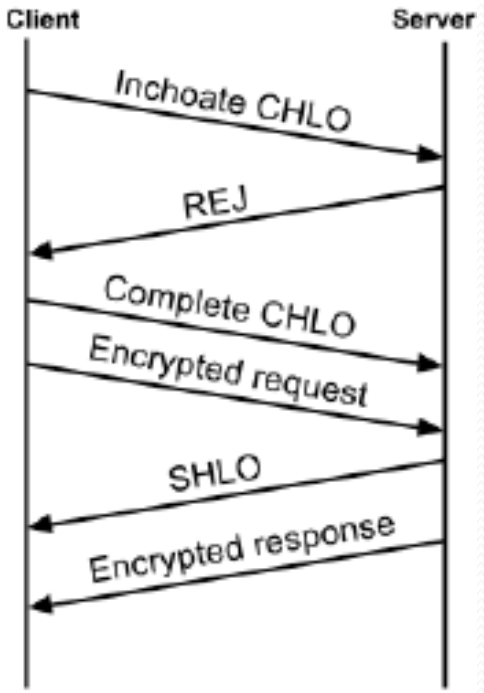
- Introduction and Motivation
- **Design and Implementation**
- Internet-scale Deployment and Performance
- Experience

Connection Establishment

- On a successful handshake, a client caches information about the origin
- On subsequent connections to the same origin, the client can establish an encrypted connection with no additional round trips
  - data can be sent immediately following the client handshake packet without waiting for a reply from the server.
  - **0-RTT handshakes**
- Use Diffie-Hellman to establish session key

Initial Handshake

- The client sends an inchoate client hello (**CHLO**) message to the server to elicit a reject (**REJ**) message, which contains
  - **Server config** with **long-term public key**
  - Certificate chain authenticating
  - Signature of the server config
  - **Source-address token**
- Client authenticates the config, sends a **complete CHLO**, containing the client's **ephemeral public key**.



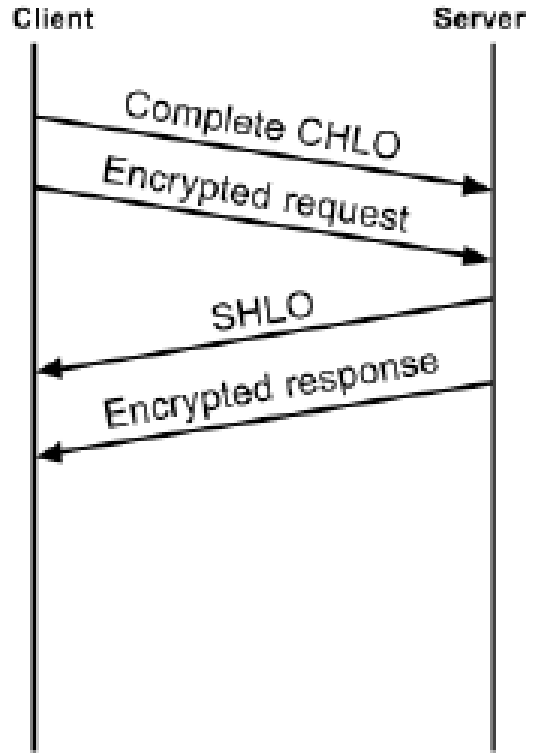
Initial 1-RTT Handshake

Initial Handshake

- Client sends application data
  - Encrypted with the **initial keys** computed from server's long-term public key and client's ephemeral private key
- Server returns a server hello (**SHLO**) message.
  - Encrypted with the **initial keys** computed from server's long-term public value and client's ephemeral public value.
  - Contain server's **ephemeral public key**
- With both peers have both sides' ephemeral public key, compute **forward-secure keys** for the connection.

Final (and repeat) Handshake

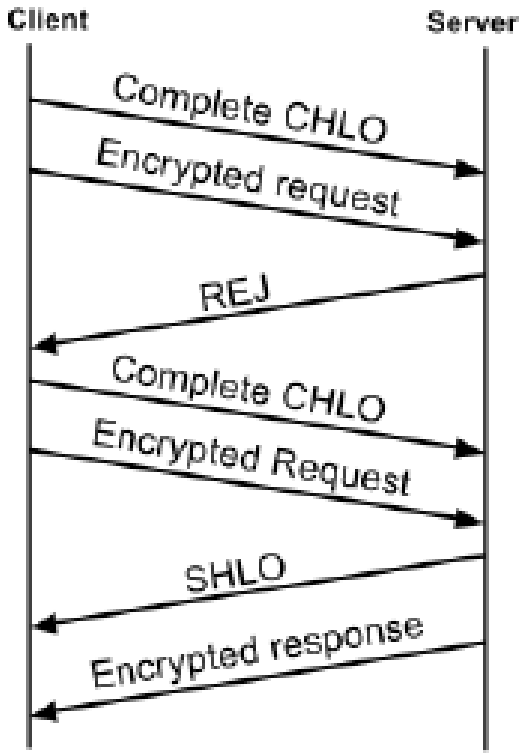
- The client caches the server config and source-address token, and on a repeat connection to the same origin, uses them to start the connection with a complete CHLO.



Successful 0-RTT Handshake

Final (and repeat) Handshake

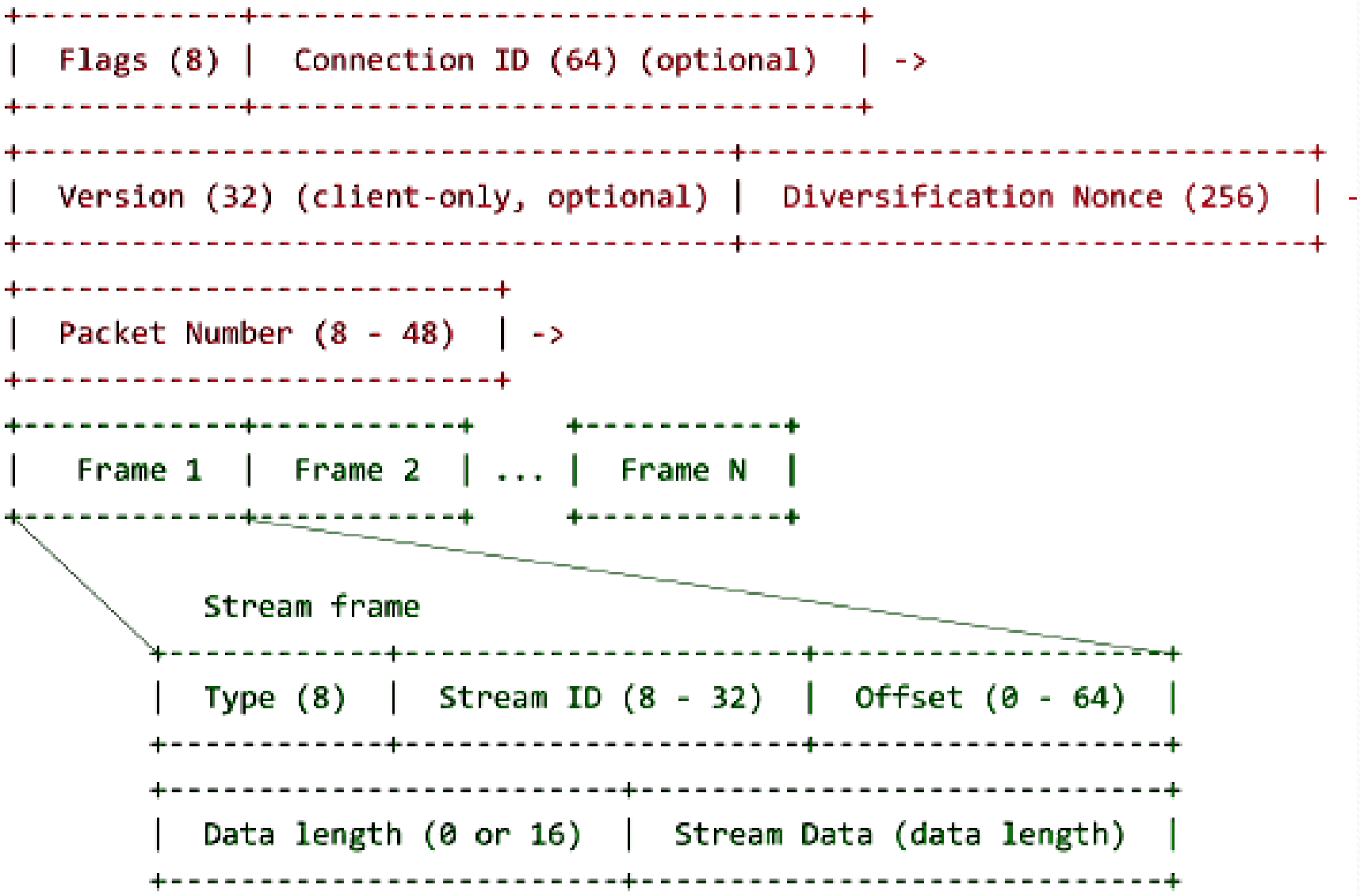
- The source address token or the server config may **expire**.
- The server replies with a REJ message, just as if the server had received an inchoate CHLO and the handshake proceeds from there.



Rejected 0-RTT Handshake

Stream Multiplexing

- QUIC supports **multiple streams within a connection**, a lost UDP packet only impacts those streams whose data was carried in that packet.
- Streams are identified by **stream IDs**
  - Stream creation is implicit when sending the first bytes with an unused stream ID,
  - Stream closing is indicated to the peer by setting a "FIN" bit on the last stream frame.
- A QUIC packet is composed of a common header followed by one or more frames


Flags (8)
Connection ID (64) (optional)
->
Version (32) (client-only, optional)
Diversification Nonce (256)
-
Packet Number (8 - 48)
->
Frame 1
Frame 2
...
Frame N
Stream frame
Type (8)
Stream ID (8 - 32)
Offset (0 - 64)
Data length (0 or 16)
Stream Data (data length)

Loss Recovery

- Each QUIC packet carries a **new packet number**, including those carrying retransmitted data.
  - Always increasing
- **Stream offsets** in stream frames are used for delivery ordering.
- Avoid retransmission ambiguity
- QUIC acknowledgments explicitly encode the delay between the receipt of a packet and its acknowledgment being sent.
  - allows for precise RTT estimation.

Others

- Congestion control
  - Cubic as in TCP
- NAT Rebinding and Connection Migration
  - QUIC connections are identified by a 64-bit Connection ID rather than 5-tuple
  - Not influenced by NAT or changes of addresses and port numbers
- QUIC Discovery for HTTPS
  - Client first sends the request over TLS/TCP.
  - QUIC server advertises QUIC support by including an "Alt-Svc" header in their HTTP responses

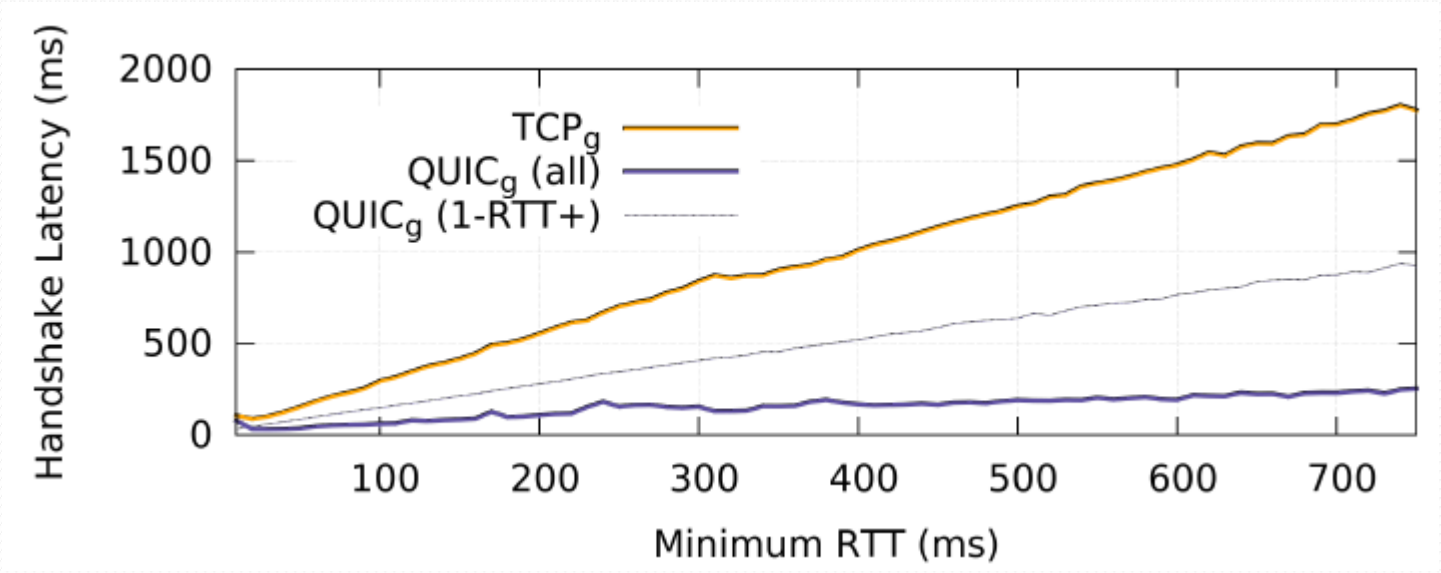
Outline

- Introduction and Motivation
- Design and Implementation
- **Internet-scale Deployment and Performance**
- Experience

Deployment and Experiment

- Implement various QUIC features in Chrome.
- Carry A/B tests
- Add QUIC support to Search and YouTube apps.
  - YouTube use HTML5, videos transferred over HTTP.
- 18-month experiments
- Compare QUIC users and TLS/TCP users on three metrics:
  - Search Latency
  - Video Playback Latency
  - Video Rebuffer Rate.

Handshake Latency



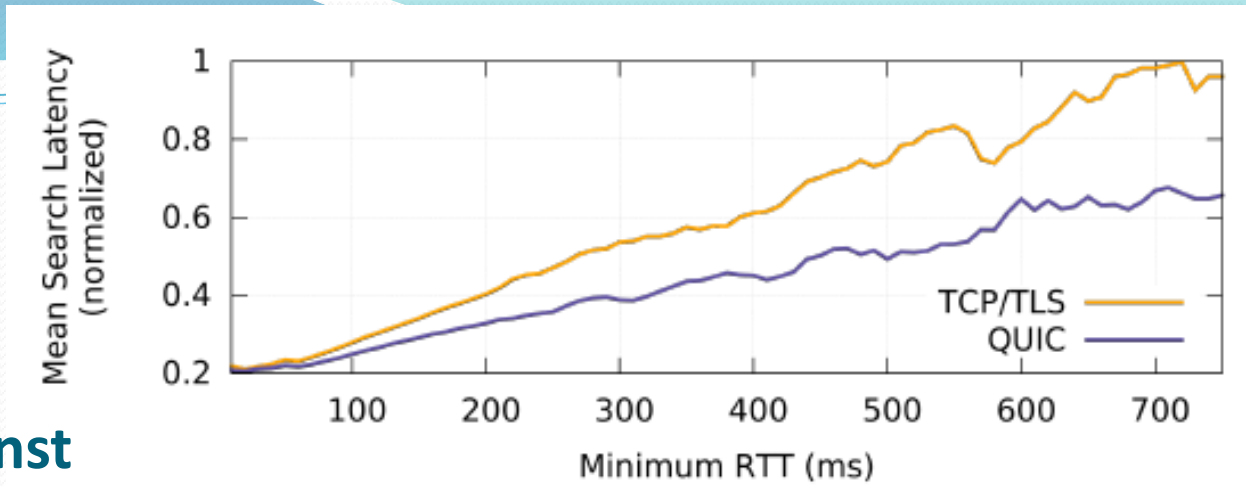
- With increasing RTT, average handshake latency for TCP/TLS trends upwards linearly, while QUIC stays almost flat.
  - due to the fixed (zero) latency cost of 0-RTT handshakes (88% for desktop and 68% for mobile)

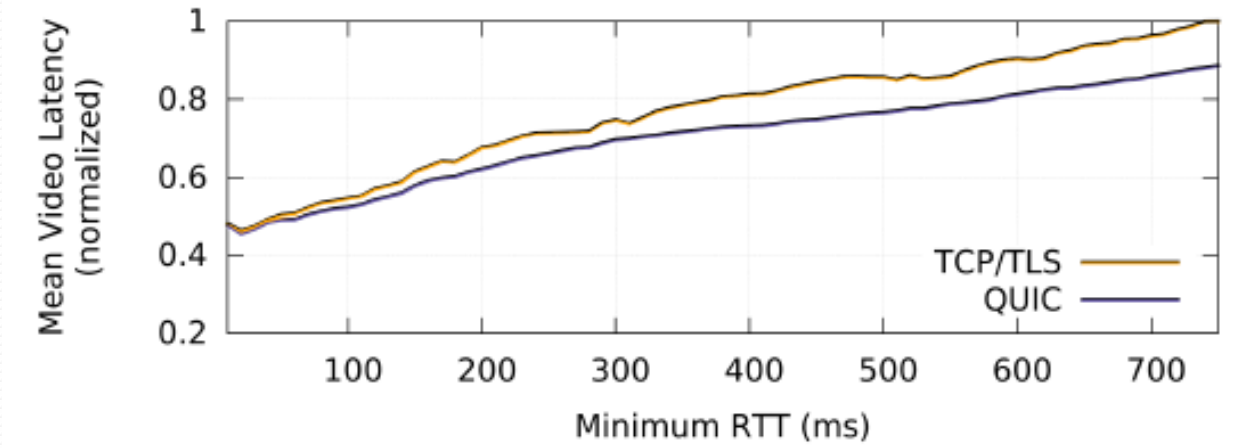
Application Metrics and Performance

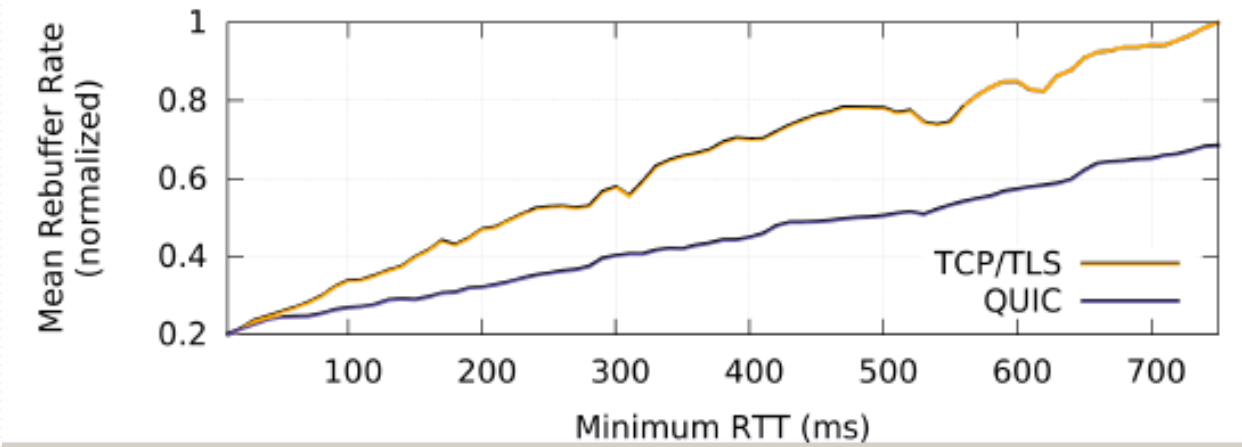
		% latency reduction by percentile						
		Lower latency				Higher latency		
	Mean	1%	5%	10%	50%	90%	95%	99%
Search								
Desktop	8.0	0.4	1.3	1.4	1.5	5.8	10.3	16.7
Mobile	3.6	-0.6	-0.3	0.3	0.5	4.5	8.8	14.3
Video								
Desktop	8.0	1.2	3.1	3.3	4.6	8.4	9.0	10.6
Mobile	5.3	0.0	0.6	0.5	1.2	4.4	5.8	7.5

		% rebuffer rate reduction by percentile				
		Fewer rebuffers				More rebuffers
	Mean	< 93%	93%	94 %	95%	99%
Desktop	18.0	*	100.0	70.4	60.0	18.5
Mobile	15.3	*	*	100.0	52.7	8.7

Performance against minimum RTTs







Mobile vs. Desktop

- 0-RTT handshakes: 88% for desktop and 68% for mobile
- First, when mobile users switch networks, their IP address changes, which invalidates the source-address token cached at the client.
  - Token also contains validated client address
- Second, different server configurations and keys are served and used across different data centers.
  - When mobile users switch networks, they may hit a different data center where the servers have a different server config than that cached at the client.

Performance By Region

- Benefits are greater in networks and regions that have higher average RTT and higher network loss.

			% Reduction in Search Latency		% Reduction in Rebuffer Rate	
Country	Mean Min RTT (ms)	Mean TCP Rtx %	Desktop	Mobile	Desktop	Mobile
South Korea	38	1	1.3	1.1	0.0	10.1
USA	50	2	3.4	2.0	4.1	12.9
India	188	8	13.2	5.5	22.1	20.2

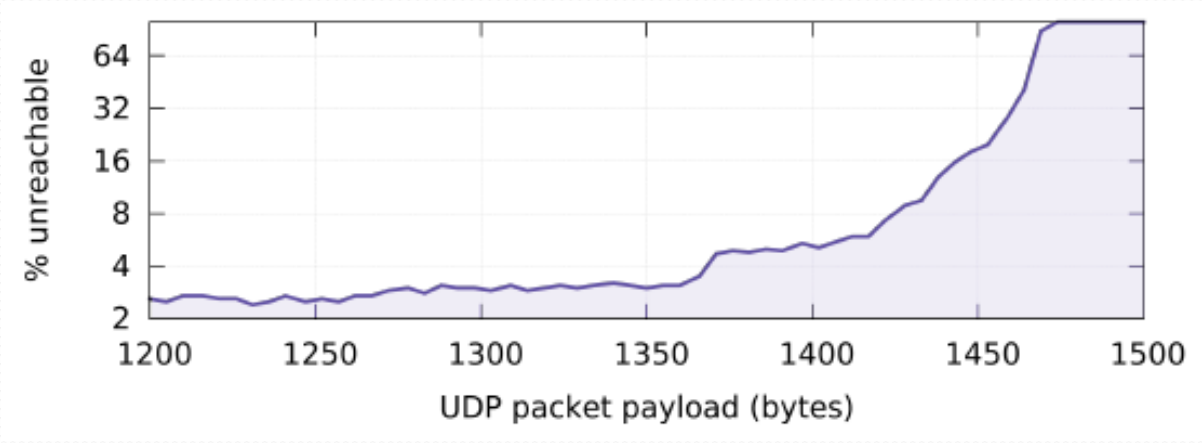
Table 3: Network characteristics of selected countries and the changes to mean Search Latency and mean Video Rebuffer Rate for users in $QUIC_g$.

Outline

- Introduction and Motivation
- Design and Implementation
- Internet-scale Deployment and Performance
- **Experience**

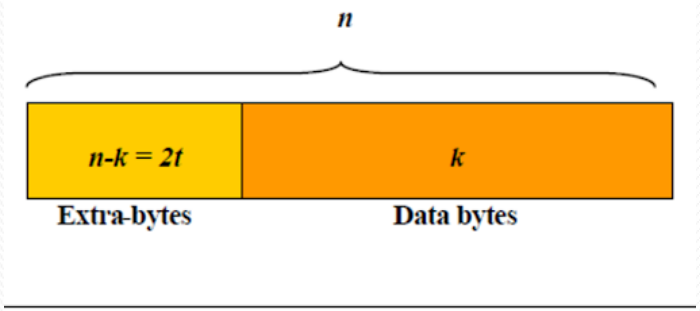
Packet Size Considerations

- performed a wide-scale reachability experiment with varying UDP payload packet size
  - Smaller UDP payload size – large overhead
  - Large UDP payload size – low reachability
- 1350 bytes by default



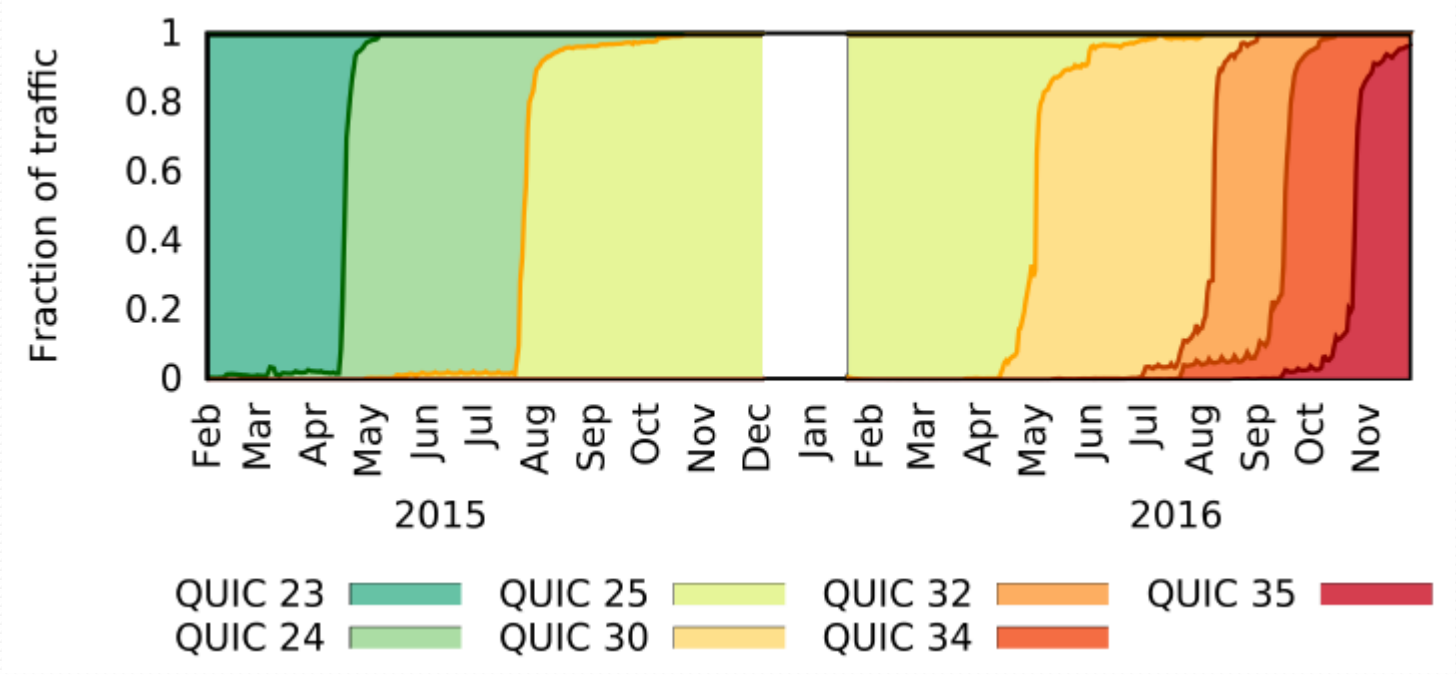
Forward Error Correction

- Use to consider using FEC to recover lost packets without an explicit retransmission.
- Experiment with various packet protection policies
  - Protecting only HTTP headers
  - Protecting all data,
- While retransmission rates decreased measurably, FEC had statistically insignificant impact on Search Latency and increased both Video Latency and Video Rebuffer Rate for video playbacks.
- Remove XOR-based FEC from QUIC



User-Space Development

- Developed in applications rather than OS kernels
- Iterate rapidly on deployment of QUIC modifications.



Review

- How QUIC reduces handshake latency?
  - 0-RTT handshake
- How QUIC reduces head of line blocking latency?
  - Multiple streams multiplex a connect.
- How QUIC avoids retransmission ambiguity?
  - Each packet has a new packet number, always increasing.
  - Loss indicated by offsets.