The QUIC Transport Protocol: Design and Internet-Scale Deployment

Published in: SIGCOMM '17

Summarized by

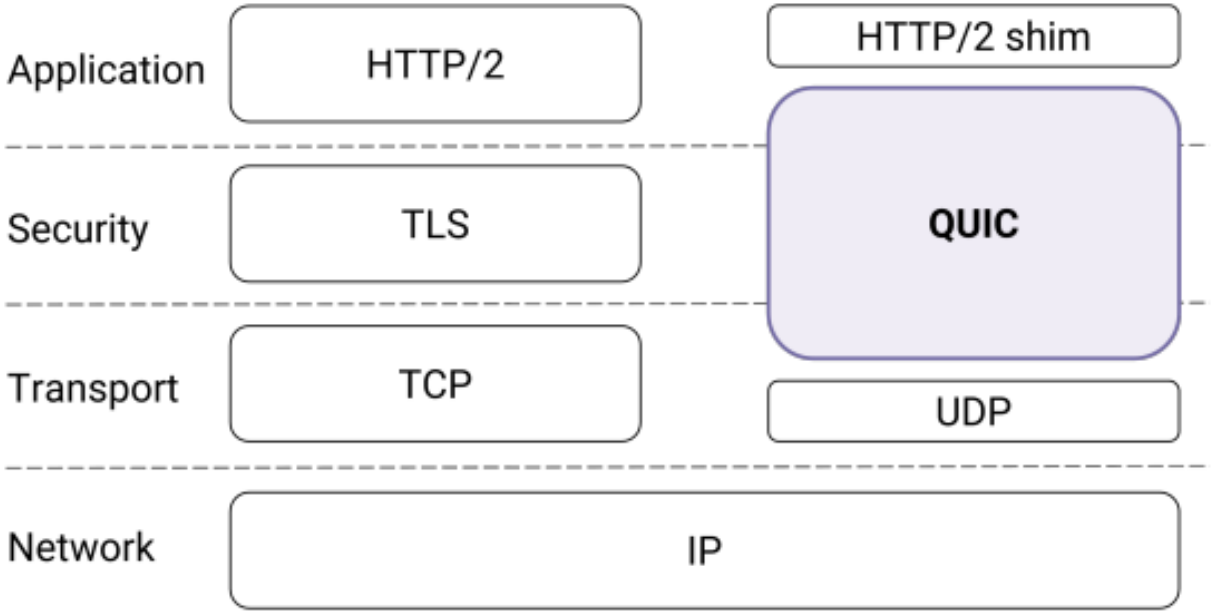
Sangwon Lim (sangwonlim@snu.ac.kr)

2023-02-07

Contents

Introduction

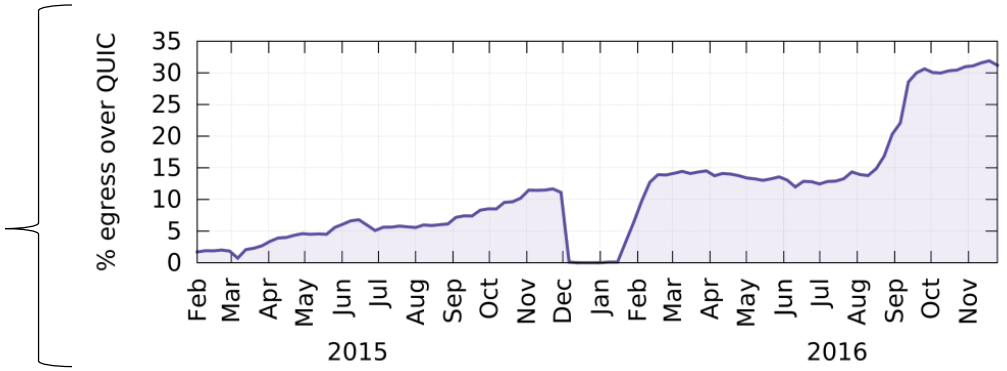
- QUIC is a new transport designed from the ground up to improve performance for HTTPS traffic



< QUIC in the traditional HTTPS stack >

Introduction

- History of QUIC
  - Protocol for HTTPS transport, deployed at Google starting 2014
    - Between Google services and Chrome / mobile apps
  - Improves application performance
    - YouTube Video Rebuffers: 15~18%
    - Google Search Latency: 3.6~8%
  - 35% of Google's egress traffic (7% of Internet)
  - IETF QUIC working group formed in Oct 2016
    - Modularize and standardize QUIC



Ref: Janardhan Iyengar's presentation

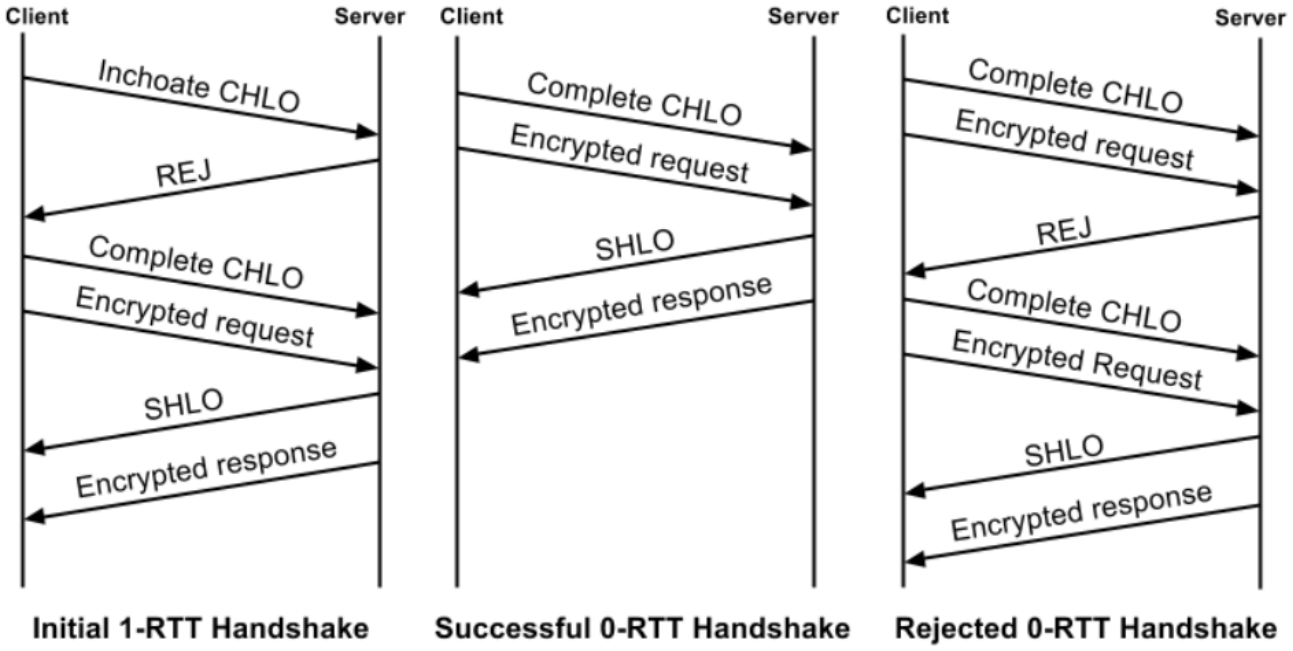
Motivation: Why QUIC??

- Growth in latency-sensitive web services and use of the web as a platform were placing unprecedented demands on reducing web latency

- Fundamental limitations of the TLS/TCP ecosystem
  - ✓ Protocol Entrenchment
  - ✓ Implementation Entrenchment
  - ✓ Handshake Delay
  - ✓ Head-of-line Blocking Delay

QUIC Design and Implementation

- Connection Establishment ← Handshake Delay
  - Mostly 0-RTT, sometimes 1-RTT



REJ message contains:

(i) a server config that includes the server's long-term Diffie-Hellman public value ,
(ii) a certificate chain authenticating the server,
(iii) a signature of the server config using the private key from the leaf certificate of the Chain, and
(iv) a source-address token: an authenticated-encryption block that contains the client's publicly visible IP address (as seen at the server) and a timestamp by the server.

QUIC Design and Implementation

- Stream Multiplexing ← Head-of-line Blocking Delay
  - Lightweight abstraction within a connection

- Authenticated and Encrypted Headers ← Protocol Entrenchment
  - Atop UDP
  - ✓Evolvability ↑

- Open-Source Implementation ← Implementation Entrenchment
  - In userspace
  - ✓Deployability ↑

QUIC Design and Implementation

- Loss Recovery
  - Unique packet number
  - Receiver timestamp
- Flow Control
  - Connection-level flow control & Stream-level flow control
  - Credit-based flow-control
- Congestion Control
  - Pluggable interface
  - Not rely on a specific congestion control

QUIC Design and Implementation

- NAT Rebinding and Connection Migration
  - 64-bit connection ID
  - Also, connection migration and multipath

- QUIC Discovery for HTTPS
  - When a QUIC client makes an HTTP request to an origin for the first time, it sends the request over TLS/TCP
  - QUIC servers advertise QUIC support by including an "Alt-Svc" header in their HTTP responses
  - On a subsequent HTTP request to the same origin, the client races a QUIC and a TLS/TCP connection

Experimentation Framework

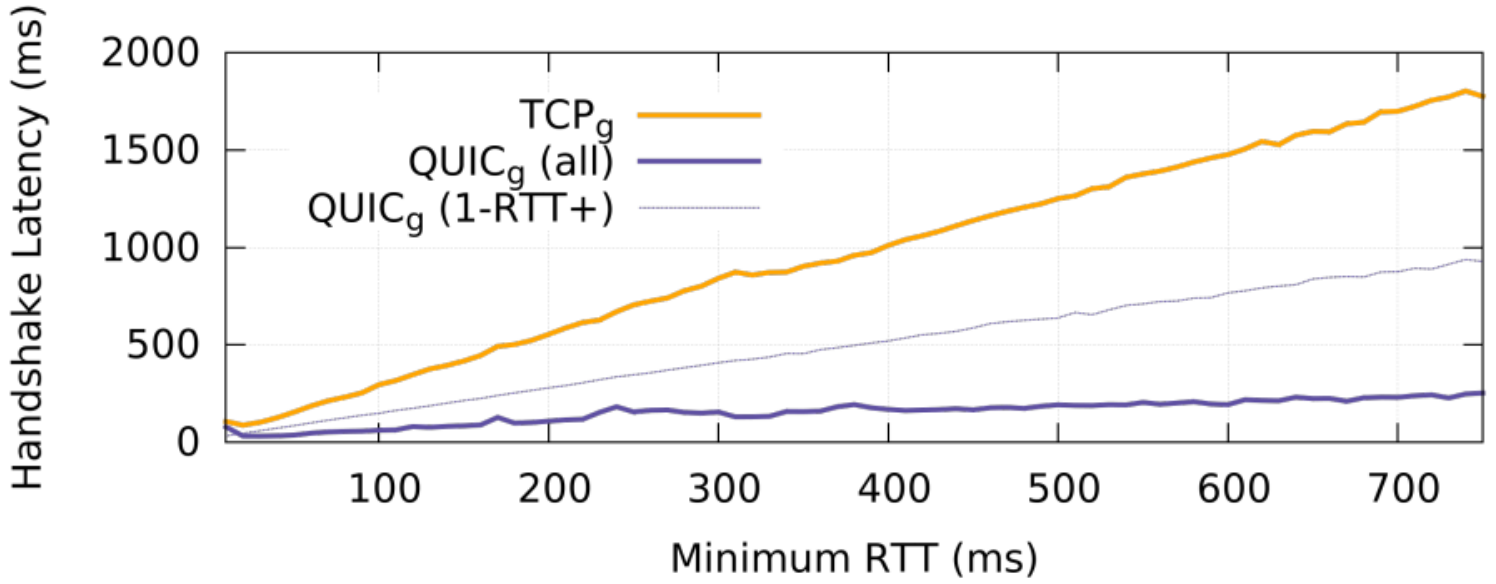
- Using Chrome
  - Randomly assign users into experiment groups
  - Experiment ID on requests to server
  - Client and sever stats tagged with experiment ID

- Novel development strategy for a transport protocol
  - The Internet as the testbed
  - Measure value before deploying any feature
  - Rapid disabling when something goes wrong

Ref: Janardhan Iyengar's presentation

QUIC Performance

- **Handshake latency**
  - at the server as the time from receiving the first TCP SYN or QUIC client hello packet to the point at which the handshake is considered complete



2-RTT + b

1-RTT + a

QUIC Performance

- **Search Latency**
  - User enter search term → entire page is loaded
- **Video playback Latency**
  - User clicks on a video → video starts playing

- Percent reduction in global Search and Video Latency for QUIC users, at the mean and at specific percentiles

		% 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

QUIC Performance

- **Video Rebuffer Rate**
  - Rebuffer time / (rebuffer time + video play time)

- Percent reduction in global Video Rebuffer Rate for QUIC users at the mean and at specific percentiles

		% 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

QUIC Performance

- QUIC improvement by Country

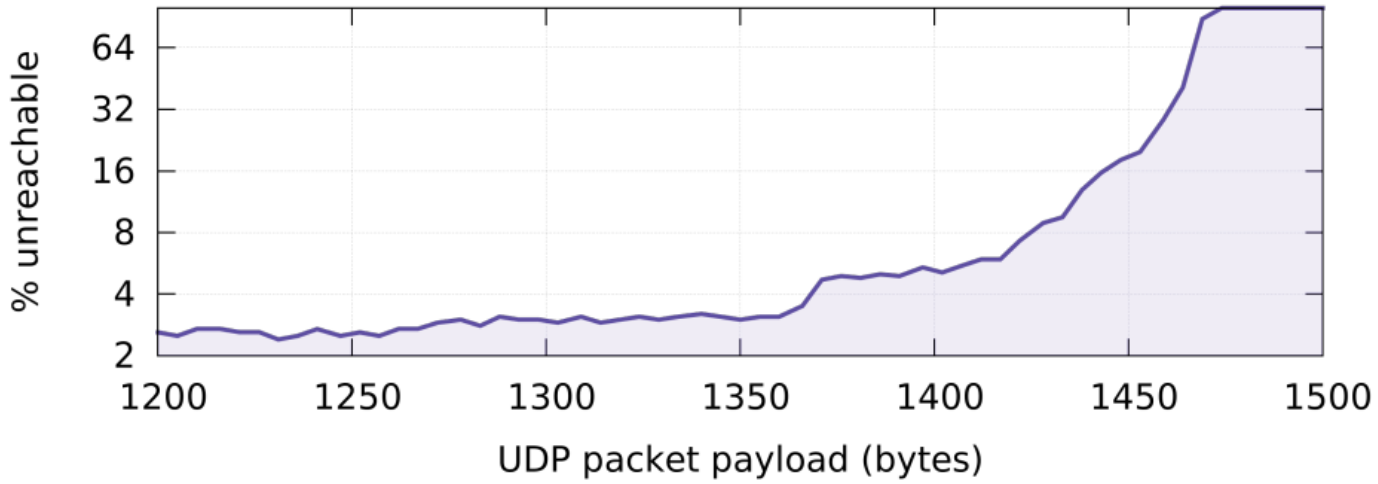
			% Reduction in Search Latency		% Reduction in Rebuffer Rate	
Country	Mean Min RTT (ms)	Mean TCP Rtx %	Desktop	Mobile	Desktop	Mobile
South Korea	38	1	1.3	1.1	0.0	10.1
USA	50	2	3.4	2.0	4.1	12.9
India	188	8	13.2	5.5	22.1	20.2

- The CPU cost of serving QUIC to approximately twice that of TLS/TCP

Experiments and Experiences

- UDP Blockage and Throttling
  - The 0.3% of users are in networks that seem to rate limit QUIC and/or UDP traffic

- Packet Size Considerations
  - Google chose 1350 bytes as the default payload size for QUIC



Experiments and Experiences

- User-space Development
  - Rapid deployment and evolution

- Middleboxes
  - Firewall used first byte of packets for QUIC classification
    - Flags byte, was 0x07 at the time
    - Broke QUIC when they flipped a bit
  - ✓ *“when deploying end-to-end changes, encryption is the only means available to ensure that bits that ought not be used by a middlebox are in fact not used by one”*

Conclusion

- QUIC was designed and launched as an experiment, and it has now become a core part of Google’s serving infrastructure
- They are working on reducing QUIC’s CPU cost at both the server and the client and in improving QUIC performance on mobile devices
- The lessons the authors learned and described in this paper are transferable to future work on Internet protocols