Taking a Long Look at QUIC

Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove
Internet connectivity is a critical service!

Why do we need yet another protocol?
Why do we need yet another protocol?

3.2 billion people with Internet access (2015)*

Internet connectivity is a critical service.

*Source: ITU and Cisco
Why do we need yet another protocol?

2 billion people with running water (2015)*

3.2 billion people with Internet access (2015)*

Internet connectivity is a critical service!

*Source: ITU and Cisco
Why do we need yet another protocol?

New techniques for more reliable and performant networks

3 billion people with running water (2015)*

3.2 billion people with Internet access (2015)*

Internet connectivity is a critical service!

*Source: ITU and Cisco
Why do we need yet another protocol?

Internet connectivity is a critical service!

3.2 billion people with Internet access (2015)*

3 billion people with running water (2015)*

New techniques for more reliable and performant networks

*Source: ITU and Cisco
Why do we need yet another protocol?

New techniques for more reliable and performant networks

3 billion people with running water (2015)*

3.2 billion people with Internet access (2015)*

Internet connectivity is a critical service*

*I.H. and Cisco (2015)
Why do we need yet another protocol?

New techniques for more reliable and performant networks

3.2 billion people with running water (2015)*

3 billion people with Internet access (2015)*

Internet connectivity is a critical service!
QUICK UDP Internet Connection

Why QUICK?
1. Facilitate rapid deployment

Why QUIC?

Quick UDP Internet Connection
1. Facilitate rapid deployment of QUIC UDP Internet Connection

Why QUIC?
1. Facilitate rapid deployment

Quick UDP Internet Connection

Why QUIC?
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes

Why QUIC?
Quick UDP Internet Connection
Why QUIC?

QUIC - Quick UDP Internet Connection

1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes

Joint study by Google and T-Mobile on YouTube’s performance over T-Mobile’s network (Velocity Conference 2014)
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve quality of experience for video
4. Does not negatively impact network traffic and decreases bufferbloat
5. Lower retransmission rates, increases throughput, Bypassing proxies

Why QUIC?

Joint study by Google and T-Mobile on YouTube's performance over T-Mobile's network (Velocity Conference 2014)

Summary Findings from Proxy Bypass

- Increases battery lifetime
- Improves quality of experience for video
- Increases throughput

Why QUIC?

Quick UDP Internet Connection

arash@thousandeyes.com
IETF 104, Mar 2019
IETF 104, Mar 2019

1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes

Why QUIC?

Cloudflare's blog, 2017

"Cloudflare ... Support TLS 1.3 by default on the server"

"Yet is middleboxes."

"The reductive answer to why TLS 1.3 hasn’t been deployed has been over a year ... and still, none of the major browsers have enabled TLS 1.3 by default."

"Cloudflare ... Cloudflare.com/why-tls-1.3-is-tls-1.3-in-browsers-yet"

"Cloudflare blog, 2017"

"Cloudflare, Inc."

"Cloudflare blog, 2017"
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes

Why QUIC?
Quick UDP Internet Connection
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes

Why QUIC?
QUICK UDP Internet Connection
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic

Why QUIC?
QUIC UDP Internet Connection
Why QUIC?
Quick UDP Internet Connection

1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic
   - Reduce handshake time (0-RTT)
Why QUIC?

1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic
   - Reduce handshake time (0-RTT)

QUICK UDP Internet Connection
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic

Why QUIC?
QUIC UDP Internet Connection

Reduce handshake time (0-RTT)
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic

- Reduce handshake time (0-RTT)

**QUIC**

**Why QUIC?**
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic

Why QUIC?

QUIC UDP Internet Connection

QUIC connection RTTs:
• 0-RTT: if keys are valid
• 1-RTT: if keys are old
• 2-RTT: if first time

Reduce handshake time (0-RTT)
1. Facilitate rapid deployment

2. Avoid ossification and meddling by middleboxes

3. Improve performance for HTTP traffic

• Prevent head-of-line blocking

• Reduce handshake time (0-RTT)

Why QUIC?

Quick UDP Internet Connection
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic

Why QUIC?
QUIC UDP Internet Connection

Why QUIC?

- Prevent head-of-line blocking
- Reduce handshake time (0-RTT)
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic
   - Prevent head-of-line blocking
   - Reduce handshake time (0-RTT)

Why QUIC?
QUIC UDP Internet Connection
HTTP/1.1 over TCP

1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic

Why QUIC?

QUICK UDP Internet Connection

QUIC?
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic
   • Prevent head-of-line blocking
   • Reduce handshake time (0-RTT)

Why QUIC?
QUIC UDP Internet Connection
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic

- Prevent head-of-line blocking
- Reduce handshake time (0-RTT)

HTTP/1.1 over TCP

QUIC UDP Internet Connection

Why QUIC?
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic

HTTP/1.1 over TCP

- Prevent head-of-line blocking
- Reduce handshake time (0-RTT)

QUIC UDP Internet Connection

Why QUIC?
HTTP/2 over TCP

- Prevent head-of-line blocking
- Reduce handshake time (0-RTT)

1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic

QUIC UDP Internet Connection

Why QUIC?
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic
   • Prevent head-of-line blocking
   • Reduce handshake time (0-RTT)

Why QUIC?
QUIC UDP Internet Connection
QUIC

Why QUIC?

1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic
   - Prevent head-of-line blocking
   - Reduce handshake time (0-RTT)
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic

Why QUIC?
QUIC UDP Internet Connection
QUICK UDP Internet Connection

Why QUICK?

1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic
   - Prevent head-of-line blocking
   - Reduce handshake time (0-RTT)
   - Improve loss recovery

QUICK UDP Internet Connection

Why QUICK?
Facilitate rapid deployment

Avoid ossification and meddling by middleboxes

Improve performance for HTTP traffic

• Modular congestion control
• Improve loss recovery
• Prevent head-of-line blocking
• Reduce handshake time (0-RTT)

Why QUIC?

QUIC UDP Internet Connection
1. Facilitate rapid deployment
2. Avoid ossification and meddling by middleboxes
3. Improve performance for HTTP traffic

Why QUIC?

QUIC: Quick UDP Internet Connection

Connection migration across IPs
Modular congestion control
Improve loss recovery
Prevent head-of-line blocking
Reduce handshake time (0-RTT)
QUIC's Timeline

Started at Google

2010s

Early 2010s

2019

IETF 104, Mar 2019
QUIC's Timeline

- Started at Google
- Early 2013
- Publicly announced
- 2019 2010s
QUIC's Timeline

- Started at Google
- Publicly announced
- IETF spec draft
- 2015
- 2013
- Early 2010s

IETF 104, Mar 2019
arash@thousandeyes.com
QUIC's Timeline

- Started at Google in the early 2010s
- Publicly announced in 2015
- IETF WG formed in 2016
- IETF spec draft published in 2016

IETF 104, Mar 2019
arash@thousandeyes.com
QUIC's Timeline

- Started at Google
- Publicly announced
- IETF spec draft
- IETF WG
- Many implementations
- Google QUIC v47
- Enroute to standardization
- (IETF QUIC)

2013
2015
2016
2019

early 2010s

IETF spec draft
IETF WG

Started at Google

Google
Cloudflare
Akamai
Fastly
Verizon
Nginx
Google's reports* QUIC's Performance Reports*
• >35% of Google's egress traffic (>7% of Internet traffic)

Google's Reports

QUIC's Performance Reports
Google's reports

• >35% of Google's egress traffic (>7% of Internet traffic)
• 3% PLT improvement on Google search

Google's Performance Reports
Google’s Performance Reports

- >35% of Google’s egress traffic (>7% of Internet traffic)
- 3% PLT improvement on Google search
- Up to 8% reduced latency on Google search

*Published at SIGCOMM 2017
Google's Reports:

- >35% of Google's egress traffic (>7% of Internet traffic)
- Up to 8% reduced latency on Google search
- Up to 18% reduced buffer time on YouTube
- 3% PLT improvement on Google search

Published at SIGCOMM 2017
Google's reports shortcomings

• Aggregated statistics

QUIC's Performance Reports
Google's reports

- Not reproducible
- Aggregated statistics

QUIC's Performance Reports shortcomings
Google's reports

- Aggregated statistics
- Not reproducible
- Limited controlled tests

QUIC's Performance Reports shortcomings
Google's reports

- Limited controlled tests
- Not reproducible
- Aggregated statistics

QUIC's Performance Reports

Shortcomings
Google’s reports shortcomings

• Limited controlled tests
• Not reproducible
• Aggregated statistics

Other QUIC evaluations

QUIC’s Performance Reports
Shortcomings

QUIC’s Performance Reports

Other QUIC Evaluations

• Limited environments/networks
• Limited controlled tests
• Not reproducible
• Aggregated statistics

Google’s Reports

Google’s reports

Other QUIC evaluations

• Limited environments/networks
• Limited controlled tests
• Not reproducible
• Aggregated statistics
Shortcomings
QUIC's Performance Reports

Other QUIC Evaluations

• Limited environments/networks
• Limited tests
• Aggregated statistics
• Not reproducible
• Limited controlled tests
Google’s reports

Other QUIC evaluations

- One old untuned version of QUIC
- Limited tests
- Limited environments/networks

QUIC’s Performance Reports

Shortcomings

- Limited controlled tests
- Not reproducible
- Aggregated statistics
- Other QUIC evaluations
Other QUIC Evaluations

• Results not statistically sound

• One old untuned version of QUIC

• Limited tests

• Limited environments/networks

Google’s Reports

Shortcomings

• Limited controlled tests

• Not reproducible

• Aggregated statistics

QUIC’s Performance Reports
Shortcomings

QUIC's Performance Reports

Google's Reports

• Limited environments/networks
• Limited tests
• One old untuned version of QUIC
• Results not statistically sound
• No root cause analysis
• Limited controlled tests
• Not reproducible
• Aggregated statistics
• Other QUIC evaluations
Our goal: provide a rigorous evaluation of QUIC

Google's reports

- One old untuned version of QUIC
- Results not statistically sound
- Limited tests
- Limited environments/networks
- Aggregated statistics

Other QUIC evaluations

- Limited controlled tests
- Not reproducible
- Aggregated statistics

Google's reports shortcomings

and how it compares to TCP

No root cause analysis

Aggregated statistics

Limited tests

Limited environments/networks

Limited controlled tests

Not reproducible

Aggregated statistics
HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP

Router (OpenWRT+TC/NETEM)

Server (Support QUIC and TCP)

Client

Compare PLTs

Comparing PLTs
HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP

1. Google servers
2. Server in Chromium

Router (OpenWRT+Tc/NetEM)
HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP

1. Google servers
2. Server in Chromium
   - Unexpected behavior
   - No control
   - No servers
HTTP Performance: QUIC vs. TCP

- Google servers
- Server in Chromium
  - Unexpected behavior
  - No control

1 - Google servers
2 - Server in Chromium

Downloading a 10MB obj at 100Mbps

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>0</th>
<th>500</th>
<th>1000</th>
<th>1500</th>
<th>2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Client (openWRT+TC/NETEM) Router

HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP

- Not performant
- Unexpected behavior
- No control

1. Google servers
2. Server in Chromium

Router (OpenWRT+TC/NETEM)

Client

Our server unadjusted

GAE

Our server adjusted

Time (ms)

0 500 1000 1500 2000

Download a 10MB obj at 100Mbps

Receive Wait
HTTP Performance: QUIC vs. TCP

Our server on EC2 (Support QUIC and TCP)

Router (OpenWRT + TC/NETEM)

Client

HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP

Our server on EC2 (Support QUIC and TCP).

Router (OpenWRT+TC/NETEM)

Client

Diagram: HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP

Our server on EC2 (Support QUIC and TCP).

Router (OpenWRT+TC/NETEM)

Client
HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP

Our server on EC2 (Support QUIC and TCP).

Router (OpenWRT+TC/NETEM)
HTTP Performance: QUIC vs. TCP

RTT = 36ms, loss = 0%

Our server on EC2 (Support QUIC and TCP)

Router (OpenWRT+TC/NETEM)

Client

Diagram showing HTTP Performance comparison between QUIC and TCP.
HTTP Performance: QUIC vs. TCP

RTT = 36ms, loss = 0%

Our server on EC2 (Support QUIC and TCP)

Router (OpenWRT+TC/NETEM)

Client
HTTP Performance: QUIC vs. TCP

RTT = 36ms, loss = 0%

Our server on EC2 (Support QUIC and TCP).

Router (OpenWRT+TC/NETEM)

Client
HTTP Performance: QUIC vs. TCP

Our server on EC2 (Support QUIC and TCP).

Router (OpenWRT+TC/NETEM)

Client

RTT = 36ms, loss = 1%

RTT = 36ms, loss = 0%
HTTP Performance: QUIC vs. TCP

RTT = 36ms, loss = 1%
HTTP Performance: QUIC vs. TCP

Our server on EC2 (Support QUIC and TCP).

Router (OpenWRT+TC/NETEM)

Client
HTTP Performance: QUIC vs. TCP

Our server on EC2 (Support QUIC and TCP).

Router

Client
HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP
HTTP Performance: QUIC vs. TCP
How much does 0-RTT help?
How much does 0-RTT help?
How much does 0-RTT help?
TCP avg. throughput: 46 Mbps
QUIC avg. throughput: 79 Mbps

Bandwidth fluctuates between 50 Mbps and 150 Mbps
Fairness
Fairness

Throughput (Mbps) vs. Time (s)

- QUIC
- TCP

Flow
- Avg. Xput
- 2.71Mbps bottleneck link, RTT=36ms

QUIC vs. TCP

Throughput:
- QUIC: 2.71 Mbps
- TCP: 1.62 Mbps
<table>
<thead>
<tr>
<th>Flow</th>
<th>Avg. Xput</th>
<th>QUIC vs. TCPx4</th>
<th>QUIC vs. TCPx2</th>
<th>QUIC vs. TCPx4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0.45</td>
<td>0.41</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.36</td>
<td>0.45</td>
<td>2.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.96</td>
<td>0.7</td>
<td>2.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.62</td>
<td>2.71</td>
<td>5Mbps bottleneck link, RTT=36ms</td>
</tr>
</tbody>
</table>

Throughput (Mbps) vs. Time (s) for QUIC vs. TCP1 and TCP2.
Under same conditions, our tests show that QUIC vs. QUIC and TCP vs. TCP resulted in fair splits of the bandwidth. 5Mbps bottleneck link, RTT=36ms.
Fairness

<table>
<thead>
<tr>
<th>Cong. Win. (KB)</th>
<th>QUIC</th>
<th>TCP1</th>
<th>TCP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.45</td>
<td>0.41</td>
<td>0.36</td>
</tr>
<tr>
<td>2.75</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.96</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.62</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.71</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Avg. Xput</th>
<th>QUIC</th>
<th>Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Fairness

<table>
<thead>
<tr>
<th>Avg. Xput</th>
<th>Flow</th>
<th>QUIC vs. TCPx2</th>
<th>QUIC vs. TCPx4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.62</td>
<td>TCP</td>
<td>0.45</td>
<td>0.36</td>
</tr>
<tr>
<td>2.71</td>
<td>TCP</td>
<td>0.41</td>
<td>0.36</td>
</tr>
<tr>
<td>2.8</td>
<td>QUIC</td>
<td>0.45</td>
<td>0.36</td>
</tr>
</tbody>
</table>

### Cong. Win. (KB)

- QUIC: [0-20]
- TCP: [0-20]

### Time (s)

- QUIC: [00-100]
- TCP: [00-100]
### Fairness

<table>
<thead>
<tr>
<th>Flow Avg. Xput</th>
<th>QUIC vs. TCPx4</th>
<th>QUIC vs. TCPx2</th>
<th>QUIC vs. TCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.45</td>
<td>TCP1</td>
<td>TCP2</td>
<td>TCP2</td>
</tr>
<tr>
<td>0.41</td>
<td>TCP1</td>
<td>TCP2</td>
<td>TCP2</td>
</tr>
<tr>
<td>0.36</td>
<td>TCP1</td>
<td>TCP2</td>
<td>TCP2</td>
</tr>
<tr>
<td>0.45</td>
<td>TCP1</td>
<td>TCP2</td>
<td>TCP2</td>
</tr>
<tr>
<td>0.75</td>
<td>QUIC</td>
<td>QUIC</td>
<td>QUIC</td>
</tr>
<tr>
<td>0.96</td>
<td>TCP1</td>
<td>TCP2</td>
<td>TCP2</td>
</tr>
<tr>
<td>0.7</td>
<td>TCP1</td>
<td>TCP2</td>
<td>TCP2</td>
</tr>
<tr>
<td>2.8</td>
<td>QUIC</td>
<td>QUIC</td>
<td>QUIC</td>
</tr>
</tbody>
</table>

5Mbps bottleneck link, RTT=36ms
HTTP Performance: QUIC vs. TCP

Desktop vs. Mobile

RTT = 36ms, loss = 0%
HTTP Performance: QUIC vs. TCP

Desktop vs. Mobile

RTT = 36ms, loss = 0%
HTTP Performance: QUIC vs. TCP

Desktop vs. Mobile

RTT = 36ms, loss = 0%

24
HTTP Performance: QUIC vs. TCP

Desktop vs. Mobile
HTTP Performance: QUIC vs. TCP

Desktop vs. Mobile

RTT = 36ms, loss = 0%
<table>
<thead>
<tr>
<th>State</th>
<th>Desktop</th>
<th>Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow</td>
<td>0.2%</td>
<td></td>
</tr>
<tr>
<td>Cong.</td>
<td>40.6%</td>
<td></td>
</tr>
<tr>
<td>App.</td>
<td>0.42%</td>
<td></td>
</tr>
<tr>
<td>Avoid.</td>
<td>91.5%</td>
<td></td>
</tr>
<tr>
<td>Limit.</td>
<td>7.1%</td>
<td></td>
</tr>
<tr>
<td>Recovery</td>
<td>0%</td>
<td></td>
</tr>
</tbody>
</table>

% of time spent in each state

HTTP Performance: QUIC vs. TCP

Desktop vs. Mobile
HTTP Performance: QUIC vs. TCP

Desktop vs. Mobile

% of time spent in each state

- Slow: 40.6%
- Cong. Avoid.: 58.8%
- App. Limit.: 0.2%
- Recovery: 0.42%

RTT = 36ms, loss = 0%
1. Evaluated an application-layer transport protocol

2. Deployed at scale with nonpublic configuration parameters.

3. Controlled experiments
   - Variety of conditions and environments

4. Instrumented the protocol
   - Multiple versions

5. Approach can be applied to future versions and protocols
   - Provided root cause analyses
   - Inferred state machines

Summary

QUIC [K]
Thank you!
<table>
<thead>
<tr>
<th>Video Qualities</th>
<th>Clients (Proxy)</th>
<th>Objects Sizes (KB)</th>
<th>Extra Loss</th>
<th>Extra Delay (RTT)</th>
<th>Rate Limits (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny, medium, high, desktop, Nexus, Moto X, QVNC, proxy, TCP</td>
<td>Proxy</td>
<td>5, 10, 100, 200, 500, 1000, 10,000, 210,000</td>
<td>0.1%, 1%</td>
<td>5, 10, 50, 100</td>
<td>Tiny, 50ms, 100ms</td>
</tr>
</tbody>
</table>

Values tested:
<table>
<thead>
<tr>
<th></th>
<th>3G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verizon</td>
<td>0.17</td>
</tr>
<tr>
<td>Sprint</td>
<td>0.31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>LTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verizon</td>
<td>2.4</td>
</tr>
<tr>
<td>Sprint</td>
<td>0.17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>3G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verizon</td>
<td>109</td>
</tr>
<tr>
<td>Sprint</td>
<td>62</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0.02</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verizon</td>
<td>0</td>
</tr>
<tr>
<td>Sprint</td>
<td>0.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1.38</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verizon</td>
<td>9</td>
</tr>
<tr>
<td>Sprint</td>
<td>62</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>55</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verizon</td>
<td>0</td>
</tr>
<tr>
<td>Sprint</td>
<td>4.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Loss (10%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verizon</td>
<td>0%</td>
</tr>
<tr>
<td>Sprint</td>
<td>0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Reordering (10%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verizon</td>
<td>0%</td>
</tr>
<tr>
<td>Sprint</td>
<td>0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Throughput (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verizon</td>
<td>0.05</td>
</tr>
<tr>
<td>Sprint</td>
<td>0.13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Throughput (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verizon</td>
<td>0.25</td>
</tr>
<tr>
<td>Sprint</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Reai Cell Networks
Proxying
Proxying
QUIC vs. TCP proxied

RTT = 36ms
loss = 0%

RTT = 112ms
loss = 0%

Proxying

% = 0%
loss = 1%
RTT = 36ms

% = 1%
loss = 0%
RTT = 36ms

% = 0%
loss = 0%
RTT = 36ms

Graphical representation of proxying.
Other Experiments

- Historical analysis of QUIC over more than a year
- Proxying
- Video streaming over QUIC
- QUIC in cellular networks
Challenges of Evaluating QUIC
Challenges of Evaluating QUIC

• Rapidly evolving
Challenges of Evaluating QUIC

• Rapidly evolving

...
Challenges of Evaluating QUIC

- No formal model for how QUIC should behave
- Gap between what is publicly released and what is deployed in production by Google (and others)
- Rapidly evolving

...