A Comparison of Network Protocols: TCP vs QUIC

CSC 466

Carl Masri
Kjalen Hansen
Jakob Roberts

QUIC - An Introduction

What is QUIC?

Simply put, QUIC (**Q**uick **U**DP **I**nternet **C**onnection) is an attempt by google to improve upon the TCP protocol. QUIC sits on top of UDP and aims to reduce connection and transport latency while maintaining the same quality of congestion control as TCP. Although it does maintain a level of security similar to TLS, it trades off security weaknesses for speed.

Limitations of Existing TCP and UDP and HTTP1/2

TCP has 2 (main) problems: **longer RTT** for connection establishment and **head of line blocking** (shown on the next slide).

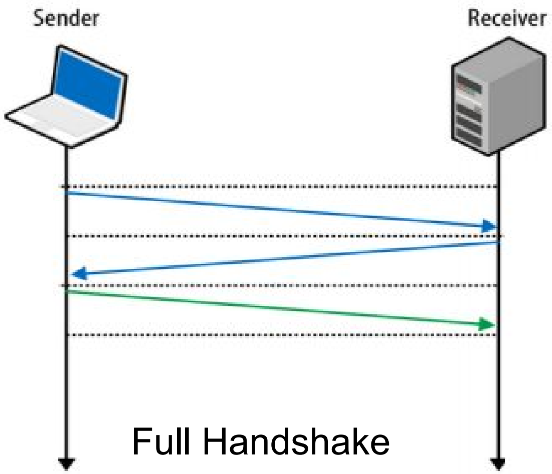
UDP has no guarantee of packet delivery and no associated encryption protocols, among the host of other issues with UDP such as vulnerabilities to being used in DOS attacks.

HTTP/1.1 has high overhead due to each request requiring a newly established connection. The addition of a security layer adds extra RTT.

HTTP/2 (based off SPDY) improves on some of HTTP/1.1's limitations, except it does nothing to improve security other than interfacing with TLS.

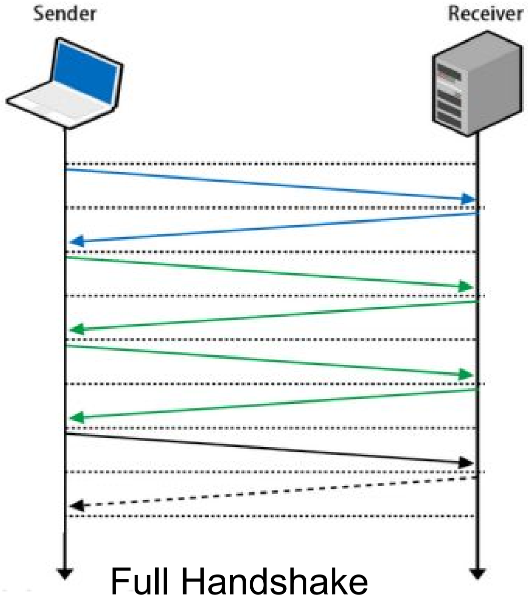
Connection requirements of QUIC compared to TCP

TCP



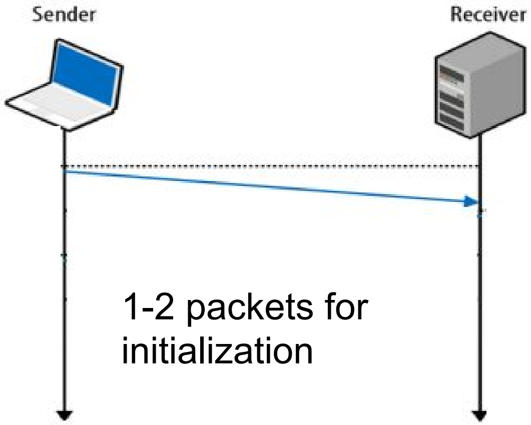
Full Handshake

TCP with TLS



Full Handshake
+TLS Authentication

QUIC



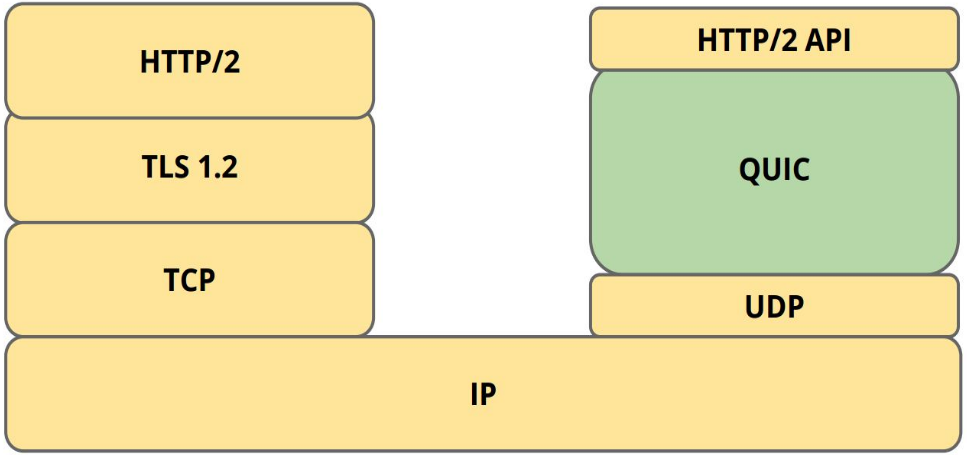
1-2 packets for
initialization

How does QUIC fit into the stack?

QUIC sits on top of UDP.

It replaces the TLS encryption with its own crypto-layer.

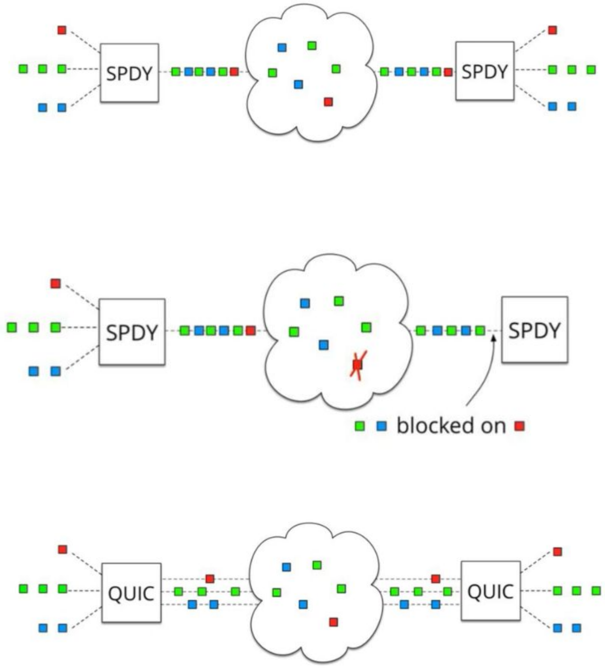
Multiplexing and connection management is handled by QUIC (only an HTTP protocol interpretation is left to the HTTP/2 API).



Head of Line Blocking

With current TCP connections, if a packet fails to transmit in the correct order, it needs to be re-transmitted and all other packets are instructed to wait until the desired one arrives.

With QUIC, this is solved with UDP as it is not dependent on the order of which packets are received. Packets can still be lost in transit but will only impact individual elements at the destination and not hold up page load time. These packets can then be retransmitted if necessary.



Forward Error Correction (FEC)

The QUIC protocol can recover from transmission errors without retransmitting lost packets. An arbitrary number of packets can be grouped together and transmitted along with one extra packet for error recovery. If a single packet of that group is lost, it can be recovered at the destination by way of Gaussian elimination. The smaller the group, the more robust against transmission errors. However, the advantages in loss recovery brought on by smaller FEC groups incur consequences of increased bandwidth overhead.

QUIC Encryption Standards

QUIC's encryption is considered to be equivalent to TLS. But, it does not stack up against some TLS modes such as TLS-DHE as the forward secrecy is lost.

The trade-offs happen from how QUIC tries to reduce latency by trading out for security weaknesses. We can take that at this point in time, you can't have good security and a fast connection.

QUIC implements two keys instead of a single as in TLS. This is the defining feature of a Quick Protocol. It has Protection against IP Spoofing and packet reordering, but it is vulnerable to connection interruptions and denial of service attacks.

Testing Methods/Methodology

We compared QUIC and HTTP/2 in both controlled and uncontrolled environments:

Controlled: Two servers were set up on a laptop: one for QUIC traffic and the other for HTTP (TCP), while the Chrome browser acted as the client. The same set of web pages were served for each, with varying number/size of embedded objects. A shaper server was used to emulate a variety of network scenarios.

Uncontrolled: Pre-defined pages were hosted on Google Sites with varying number/size of objects. Pages were loaded on Chrome over a standard home WiFi connection. The Netem command was utilized to add delay and loss constraints.

In both scenarios, data was collected using a custom-built script which automatically captured the Page Load Time (PLT) for each page.

Results:

Controlled Scenario

🟩 = Faster

🟥 = Slower

🟪 = Similar

# Objects	2, 64, 128
Loss (%)	0, 2
RTT (ms)	5, 100, 200

TCP /w SSL	128 @ 300Kb	64 @ 300Kb	2 @ 300Kb	128 @ 10Kb	64 @ 10Kb	2 @ 10Kb	128 @ 1Kb	64 @ 1Kb	2 @ 1Kb	128 @ 100bytes	64 @ 100bytes	2 @ 100bytes
0% (5ms)	🟥 605.00	🟥 323.14	🟩 65.84	🟩 199.03	🟩 129.51	🟥 49.28	🟥 158.79	🟩 111.79	🟩 50.58	🟥 184.03	🟥 103.12	🟥 50.64
2% (100ms)	🟩 3275.24	🟩 2431.26	🟩 817.54	🟩 1199.27	🟩 937.58	🟩 621.54	🟩 1029.49	🟩 826.91	🟩 625.82	🟥 1169.69	🟥 808.51	🟩 435.27
2% (200ms)	🟩 8672.42	🟩 5237.49	🟩 1642.48	🟩 2096.03	🟩 1836.93	🟩 1309.05	🟩 2004.88	🟩 1463.51	🟩 1228.80	🟥 2046.25	🟥 1544.36	🟩 1143.90

QUIC	128 @ 300Kb	64 @ 300Kb	2 @ 300Kb	128 @ 10Kb	64 @ 10Kb	2 @ 10Kb	128 @ 1Kb	64 @ 1Kb	2 @ 1Kb	128 @ 100bytes	64 @ 100bytes	2 @ 100bytes
0% (5ms)	🟩 320.00	🟩 200.00	🟥 120.00	🟥 351.52	🟥 192.54	🟩 20.76	🟩 105.00	🟥 115.00	🟥 65.00	🟩 68.00	🟩 43.00	🟩 38.00
2% (100ms)	🟥 6537.42	🟥 4531.96	🟥 1550.00	🟥 2350.00	🟥 1520.85	🟥 1275.34	🟥 1827.38	🟥 1234.75	🟥 813.75	🟩 642.54	🟩 592.37	🟥 538.43
2% (200ms)	🟥 15875.84	🟥 11243.37	🟥 3175.85	🟥 4195.46	🟥 3376.52	🟥 1834.75	🟥 3957.52	🟥 2575.86	🟥 1386.45	🟩 821.37	🟩 759.38	🟥 1754.39

Results:

Uncontrolled Scenario

HTTP/2	high-small	high-medium	high-large	low-small	low-medium	low-large
base (25 RTT)	🟥 2245.57	🟥 2806.25	🟥 3893.67	🟪 1562.48	🟪 1469.96	🟥 1600.41
L0 (+50 RTT)	🟥 5053.31	🟥 6134.58	🟥 6394.57	🟩 2592.05	🟪 2654.14	🟩 2732.70
L0 (+100 RTT)	🟥 7847.94	🟥 9021.47	🟥 9459.26	🟩 3722.86	🟩 3853.59	🟪 4089.17
L0 (+200 RTT)	🟩 5918.65	🟩 5980.61	🟩 6145.37	🟩 5963.7	🟩 5954.01	🟩 5924.07
L2 (25 RTT)	🟥 2648.72	🟥 3343.60	🟥 3877.77	🟥 1647.18	🟥 1793.56	🟥 2019.36
L2 (+50 RTT)	🟥 5796.23	🟥 6677.43	🟥 7128.77	🟪 2683.16	🟥 3015.96	🟥 2898.59
L2 (+100 RTT)	🟥 9326.96	🟥 10713.48	🟥 10821.75	🟩 3662.97	🟩 3997.66	🟥 4324.80
L2 (+200 RTT)	🟥 15516.48	🟥 18011.45	🟥 18726.92	🟥 6494.386	🟥 7172.89	🟥 7173.98

🟩 = Faster

🟥 = Slower

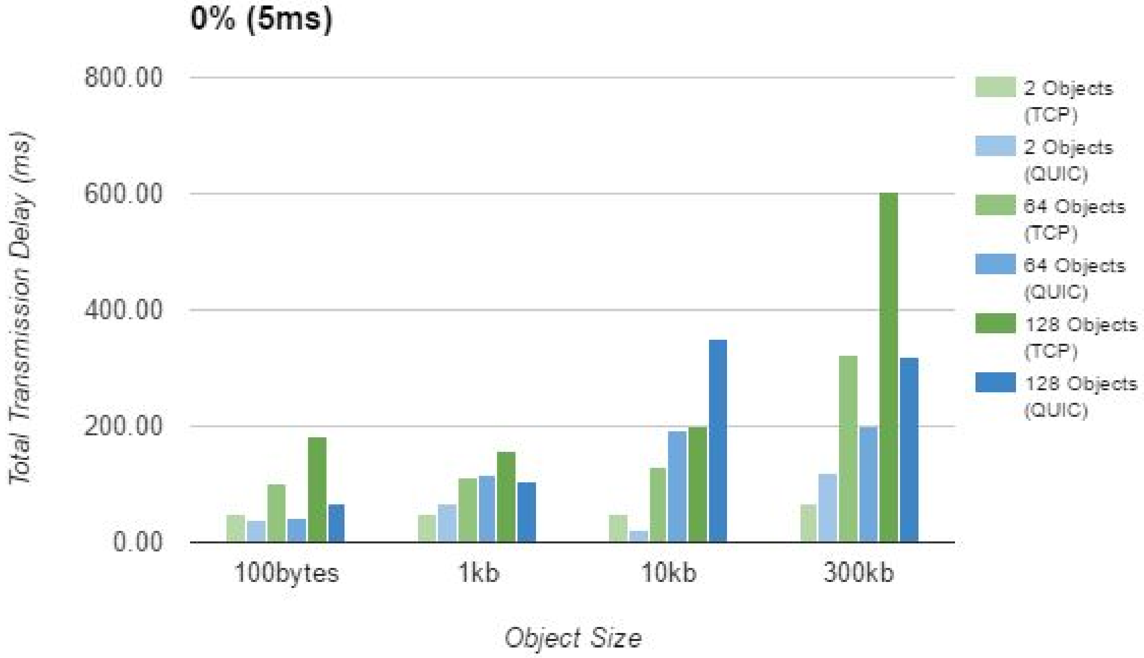
🟪 = Similar

QUIC	high-small	high-medium	high-large	low-small	low-medium	low-large
base (25 RTT)	🟩 1580.50	🟩 2327.54	🟩 3237.99	🟪 1581.93	🟪 1468.94	🟩 1493.72
L0 (+50 RTT)	🟩 2686.37	🟩 3455.32	🟩 4032.86	🟥 2685.00	🟪 2628.32	🟥 2864.02
L0 (+100 RTT)	🟩 3932.44	🟩 4368.92	🟩 5067.29	🟥 3879.88	🟥 4146.05	🟪 4052.40
L0 (+200 RTT)	🟥 6476.77	🟥 7146.86	🟥 8034.79	🟥 6330.42	🟥 6570.78	🟥 6429.07
L2 (25 RTT)	🟩 1704.97	🟩 2741.44	🟩 3025.71	🟩 1575.11	🟩 1647.19	🟩 1835.15
L2 (+50 RTT)	🟩 2984.44	🟩 3394.49	🟩 4260.49	🟪 2629.70	🟩 2861.70	🟩 2795.80
L2 (+100 RTT)	🟩 4071.21	🟩 5380.59	🟩 5153.69	🟥 3937.77	🟥 4051.51	🟩 4127.04
L2 (+200 RTT)	🟩 8133.15	🟩 7538.29	🟩 9122.05	🟩 6391.33	🟩 6735.88	🟩 6897.86

# Objects	10, 196
Loss (%)	0, 2
RTT (ms)	25,75,125,225

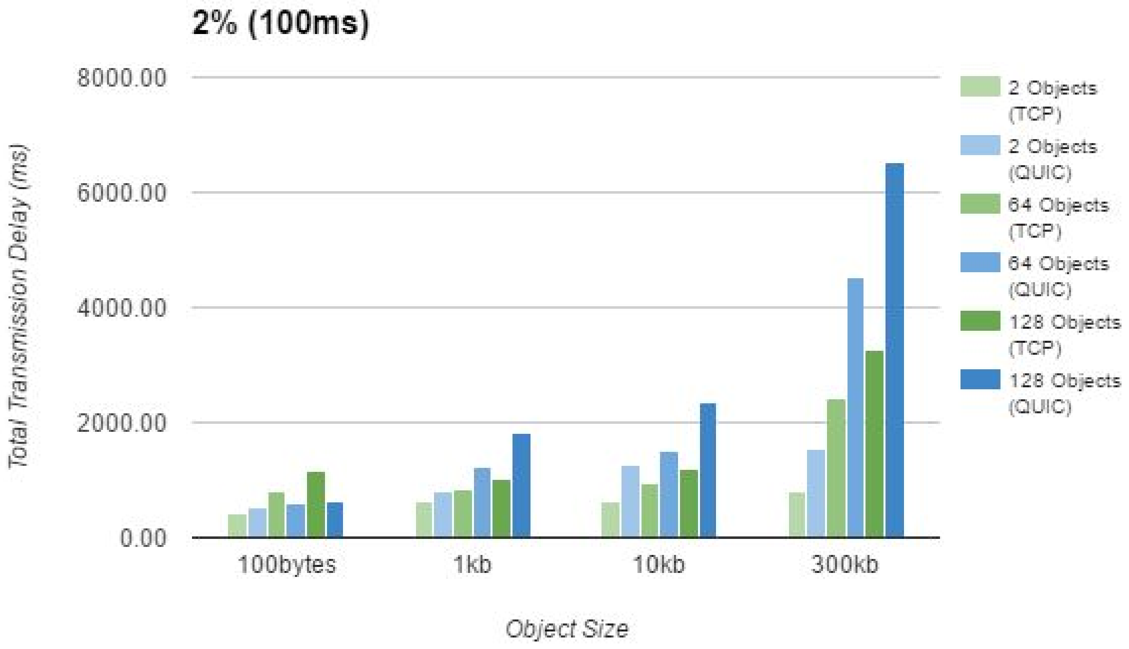
Analysis of Results:

Controlled Scenario pt1 - 0% Loss, 5ms



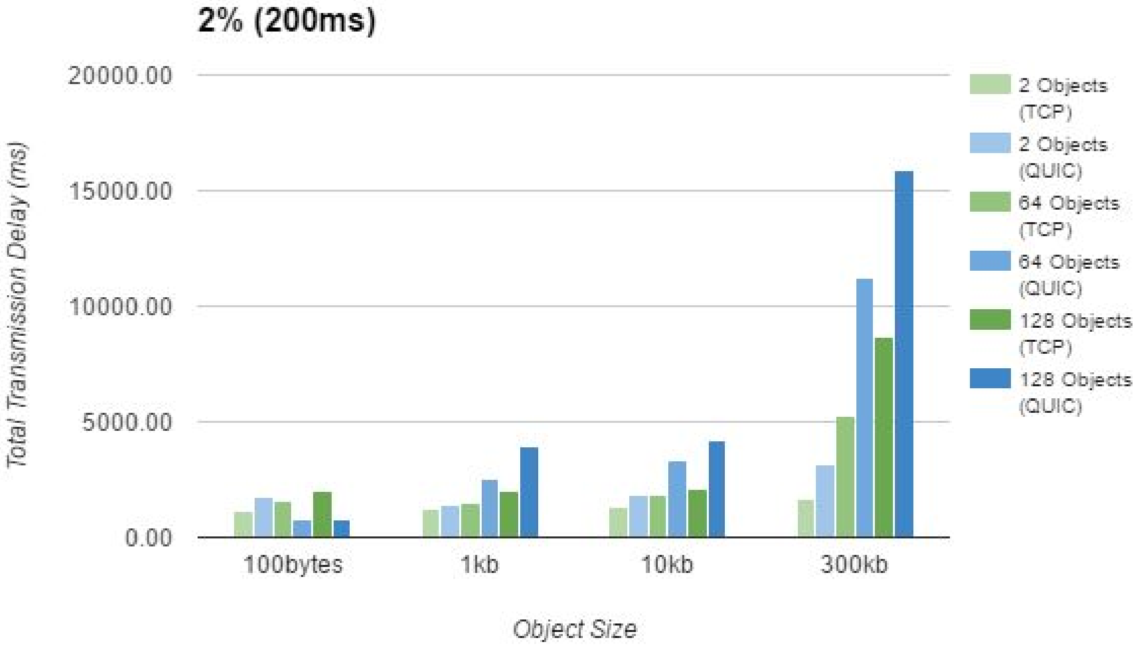
Analysis of Results:

Controlled Scenario pt2 - 2% Loss, 100ms



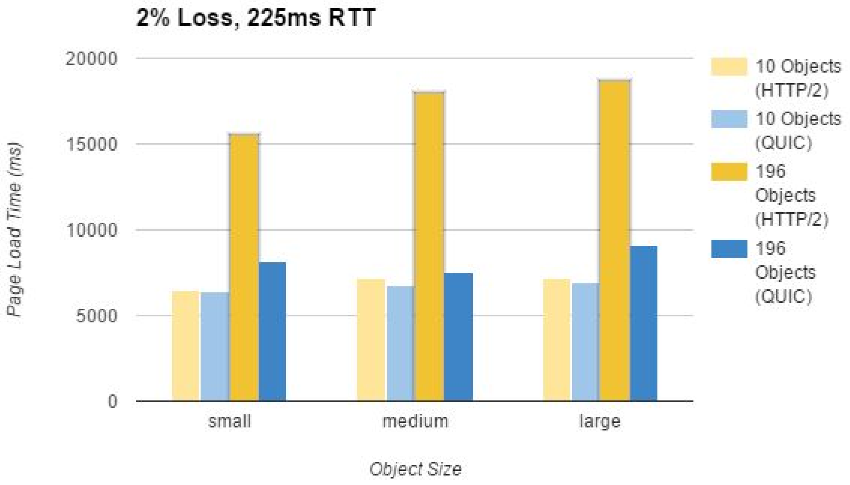
Analysis of Results:

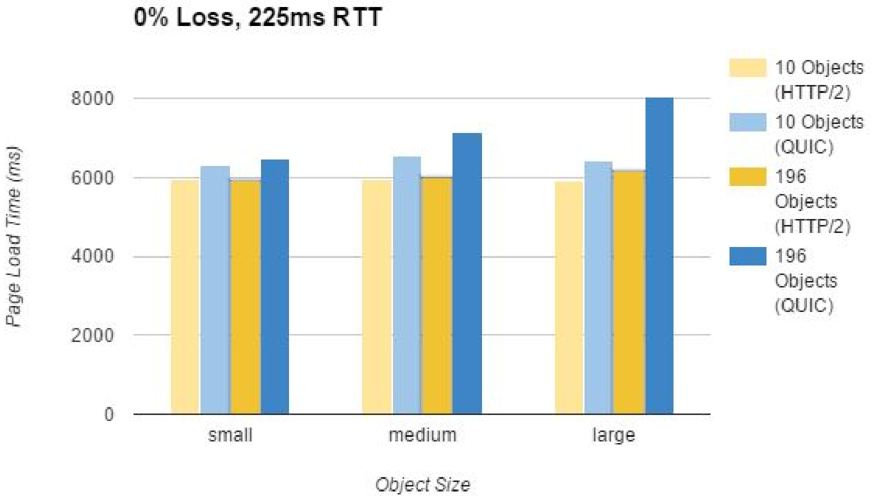
Controlled Scenario pt3 - 2% Loss, 200ms



Analysis of Results:

Uncontrolled Scenario pt1



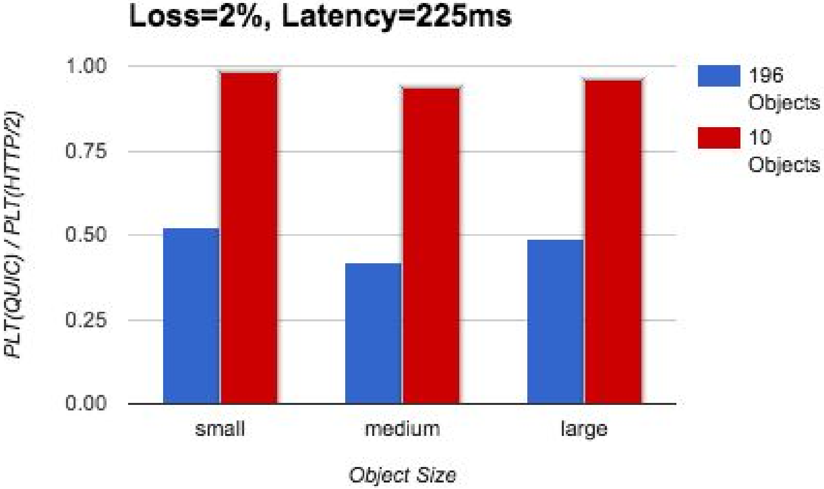


HTTP/2 really suffers in high loss and high latency scenarios when downloading a large number of objects

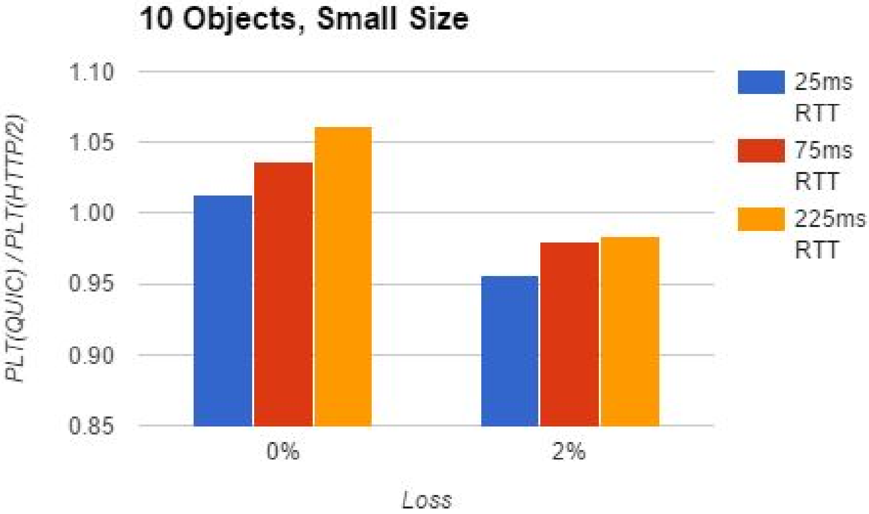
However, QUIC performs worse than HTTP/2 in a similar scenario but with no additional packet loss

Analysis of Results:

Uncontrolled Scenario pt2



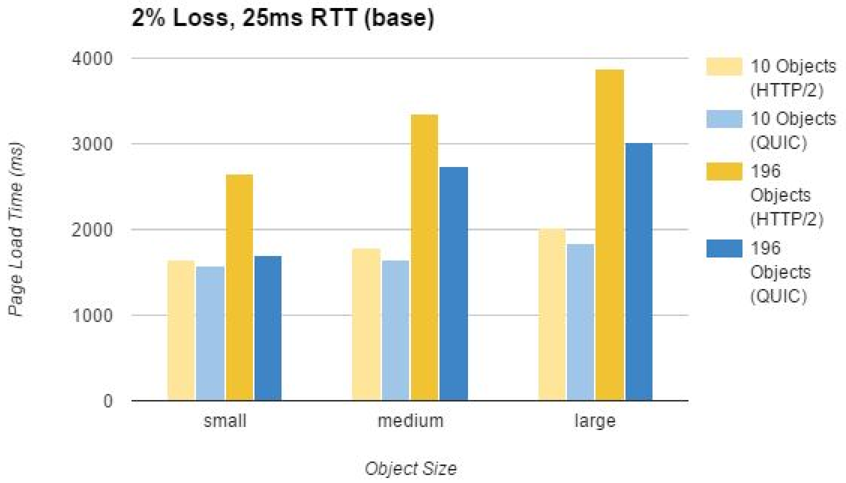
QUIC outperforms HTTP/2 with a higher object quantity.



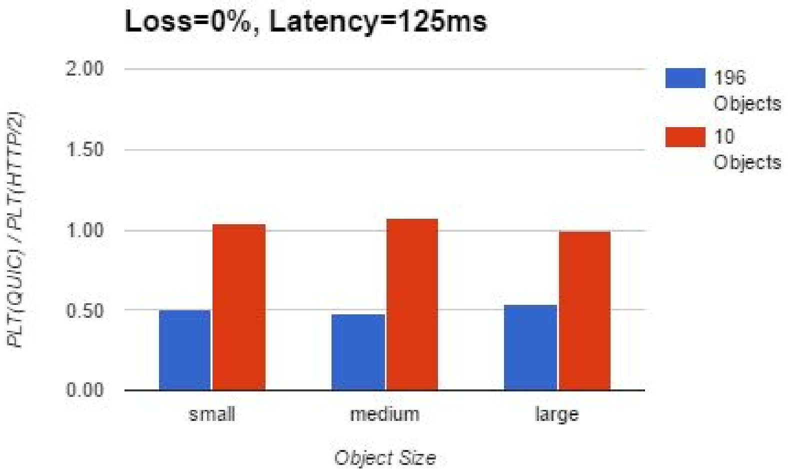
HTTP/2 performs better in a lossless scenario with a small quantity of objects, but once loss is introduced QUIC performs better.

Analysis of Results:

Uncontrolled Scenario pt3



QUIC outperforms HTTP/2 when there is no added latency but high loss.



HTTP/2 is a bit faster, but comparable to QUIC when there are few objects. When the object quantity increases, QUIC is about 2 times faster than HTTP/2.

How we could further improve testing

The current mechanisms for collecting PLT data aren't the greatest. To improve on these, we could build our own browser plugin which analyzes the network activity to a higher level of detail (takes into account queuing time, etc).

The controlled testing environment could be expanded to include embedded objects which are fetched from a separate domain.

The toy QUIC client made available for integration testing by Google does not lend itself well for testing. The server is extremely basic and required a lot of work to make usable as a test platform. We expect that as the QUIC protocol matures, so will the available tools.

Personal View of QUIC

QUIC has some really interesting concepts, and given Google's success with the protocol it's likely we'll see a few of these improvements in future versions of TCP. QUIC is not planned to be a replacement for TCP, but rather help guide its evolution forward. It is likely the next big overhaul of TCP will have QUIC elements, much like how SPDY helped the creation of HTTP/2.

QUIC has encryption built in. It is a fairly good level of encryption but future versions of TLS will be incorporated into QUIC.