Project Report, 03 Dec 2019

A comparative study of HTTP/3 -

A reliable congestion aware application protocol over UDP!

Pratyush Ranjan, pranjan@cs.stonybrook.edu

Publicly available @ https://github.com/prtsh/Quic-vs-Spdy

1. Introduction

The projects studies the performance of HTTP/3 over real world scenario. With the recent introduction of Quic, this project helps in quantifying HTTP/3 performance, as page load time, with respect to HTTP/2*. The project partially emulates and follows the paper ***"How Quic is Quic?"*** [4]. The experimental setup, metrics measurement and the goal of the project are described in subsequent section.

HTTP/3-over-Quic or *HTTP-over-Quic* or *HTTP/3* is a new application layer protocol currently under development and standardization. The IETF QUIC protocol architecture has two parts and these must be distinguished and studied separately:

1. The ***transport QUIC***, a generic transport layer to support protocols including but not limited to HTTP, the focus is only on the transport layer standardization.
2. The "***HTTP over QUIC***" a.k.a. HTTP/3 layer, which focuses on the HTTP and Quic interlinking; serving data over multiplexed HTTP streams with Quic layer managing the traffic flow and congestion control, with the base transport layer for the http streams as a UDP.

The working documents and the IETF drafts have been generously made available at [8].

Quic solves the bottleneck with the previous and current generations of HTTP over TCP in the following ways-

1. No TCP, No HOL, No problem.

Unlike HTTP1.1, which provided parallel connections and pipelining, HTTP2 used multiplexing. This shifted the HOL blocking from the application layer to the TCP layer. Quic is a multiplexing protocol, using UDP as a transport layer to handle the application layer streams. Since order is irrelevant in the UDP and the datagrams are not windowed, there is no HOL blocking in transport layer. The Quic application layer is intelligent enough to reorder the requests and decide the priorities. But this does not solve the problem of HOL at lower levels such as at router's queue.

2. Pacing the packets, Dynamically.

Apart from the conventional traffic congestion control, Quic provides packet pacing; packet request rate is adjusted as runtime depending on the latency or delay between the packet sent and received. This technique is implemented on top of the retransmission and fallback methods used in the congestion control and it is not dependent on the packet loss. This allows Quic to be much more adaptive to the turbulent networks and adjust the packet rate depending on the network speed and is expected to improve user experience.

3. Network Can change, Stream remain.

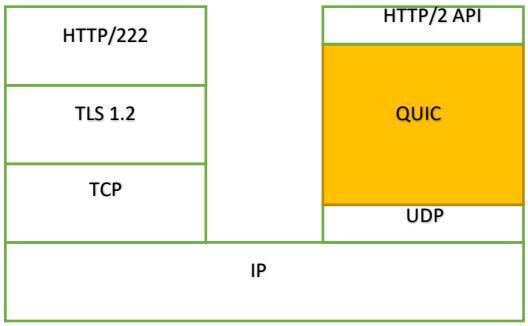
TCP/IP uses IP+port source/destination quartet to identify and multiple a connection. Quic uses a unique identifier to mark a stream (on UDP). Each client-server connection is a stream with requests multiplexed and identified by an ID. A server can identify the stream using this unique identifier. Whenever a network changes, for an instance when a client moves from a wifi to cellular, the stream identifier helps the client-server to make sure the connection persists. If it were HTTP/2, the connection would be reset and a new TCP connection would be required for continuous connectivity.

***4.* Save Packet retransmission, Use Error Correction.**

Forward Error Correction or FEC helps particularly in case of high loss network. It allows a reliable data connection by reducing the amount of retransmission required in case of packet loss.

*HTTP/2 here is standard Application layer protocol, published as RFC7540 in 2015. It evolved from the SPDY protocol developed at Google in 2008/9.

Where it all fits?



2. Problem Statement

Page load time is the most important metric performance metric and affects user experience and behaviour, such as site visits and returns. PLT is directly dependent on the application layer protocol, and is often used as a metric to compare these protocols.

HTTP 1.0 is a one-tcp per connection non-persistent model and was replaced by HTTP1.1, the persistent-tcp with parallel connection model. HTTP1.1 which was replaced by a newer multiplexing model HTTP2.0. Currently HTTP3 or HTTP-over-Quic, with features as discussed in the previous section, is being developed as a replacement to HTTP2.0. The goal of this project is to measure the performance of Quic with respect to HTTP2.

2.1. HTTP/3 availability state as of late 2019

In a recent survey [11, 12], Quick use share has been shown to be [14]

1. 10% of downlink traffic and 4% of uplink traffic, by a major broadband service provider.
2. somewhere around 50% of youtube egress volume,
3. around 80/90% for a version of facebook app, another 10-20% is http2 for A/B testing.
4. Geolocation : 20% in China, users on Android, (Akamai CDN data)
5. Most User applications don't have Quic enabled by default.

The traffic share depends on the vantage point and penetration of google services. Most of the share is from google services. Following are the two major challenges with Quic:

1. Quic is not easy and the protocol. It's dynamic and has been constantly changing since inception. It also implies that most implementation on github are adhoc and are not true-to-spec.
2. There are not many early adopters and the true working protocol is confined mainly among the google services.

Quic implementations are available on github [9]. There are IETF versions of QUIC Transport as well. Google implemented the protocol and subsequently deployed it both in their widely used browser (Chrome) and in their services such as search, you-tube and gmail. I have decide to use most of the google based services and google software (browser) in this experiments.

2.2. Clients

There is no vendor in the market currently with a working IETF HTTP/3. Google has shipped Chome browser with a working implementation of Google's own QUIC version, but that is not a standard implementation and does not interoperate with the IETF QUIC protocol. Google's HTTP implementation is different from the HTTP/3.

2.3. Servers

1. Nginx server[10], about 67% is done as of Nov 2019, part of nginx 1.17, no definite timeline.
2. As of Oct 2019, Apache has not given any timeline or roadmap on Quic support.

3. *Solution*

In order to benchmark Quic, I developed four websites of different page size and count, install client(chrome), a traffic shaper, and scripts to automate the HAR log capturing and parsing. Finally, results are displayed in section 4. The experiment and methods are detailed in subsequent sections.

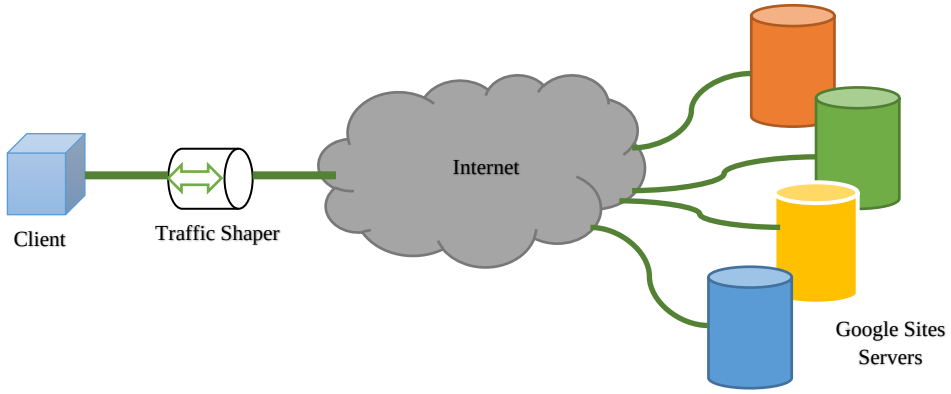
3.1. *Experiment and* Testbench Setup

There are many ways for setting up the testbench. Following methods could be used to experiment with the synthetic pages over a controlled environment:

1. Using mininet on LAN, similar to [2], this is highly synthetic and requires varying network delay or simulating packet losses to get close to a real world traffic. I am not sure how realistic it is.
2. Using a local server on WAN, downloading the static pages of common websites locally, enabling Quic in chromium and grading Quic by downloading the pages on a local client. Similar to [3]. This is closer to reality - how a user may use a website?

Between 2016 and 2018, the benchmarking area has witnessed a copious amount of work on synthetic setup such as in [2,3,5]. These experiments deploy a controlled network, with preconfigured server and clients and static pages. In this project, I will be experimenting with real web pages as in [4]. Experimenting with real pages is challenging because either most of the popular Alexa indexed websites do not support Http/Quic, or, there is no standard IETF-Quic which can be used to validate a server proclaiming as HTTP/3 compliant. The testbench consists of four components and are linked as below:

Testbench Setup



Various components are as following :

1. Client/Local machine – a regular laptop with Chrome browser with Http/3 enabled running on a mac-os.
2. Network traffic shaper – I spent some time to find out an easy-to-use and reliable network shaper for Mac and came up with ***dnctl and pfctl.***
3. Google sites – the ingenious solution to a lack of test websites is to use Google sites. Google sites are readymade easy-to-use web servers which could be modified to include different types and quantities of objects such as media, java-scripts, images or videos.
4. Chrome HAR capturer [7] for capturing the webpage log, it contained information such as page load time, urls, object type, size of objects etc.

I am using a regular laptop (mac-os) with Chrome browser to download the website contents and Chrome HAR capturer [7] to automate this process and generate log files in HTTP Archive (HAR) format. The capturer uses chrome debug protocol to instrument chrome. HAR files contain every necessary information to extract page load times from the measurements. Also, browser caching was turned off during the process (as this would result in a highly biased output, independent of the protocol used.)

3.2. HW and SW specification

System - Mac-book Air 2015, Intel core i5 1.6Ghz, 8GB RAM, 128GB SSD.
Chrome version – Version 78.0.3904.108 (Official Build) (64-bit)
MacOS version – 10.14.6 (Mojave) (64-bit)

3.3. Experiment Design

Quic and HTTP2 both use multiplexing to speed up the traffic and for maximum utilization of transport protocol. They both could be best compared for websites with large number of objects (irrespective of object sizes). Keeping this in mind, there are following types of webpages, similar to the paper[4].

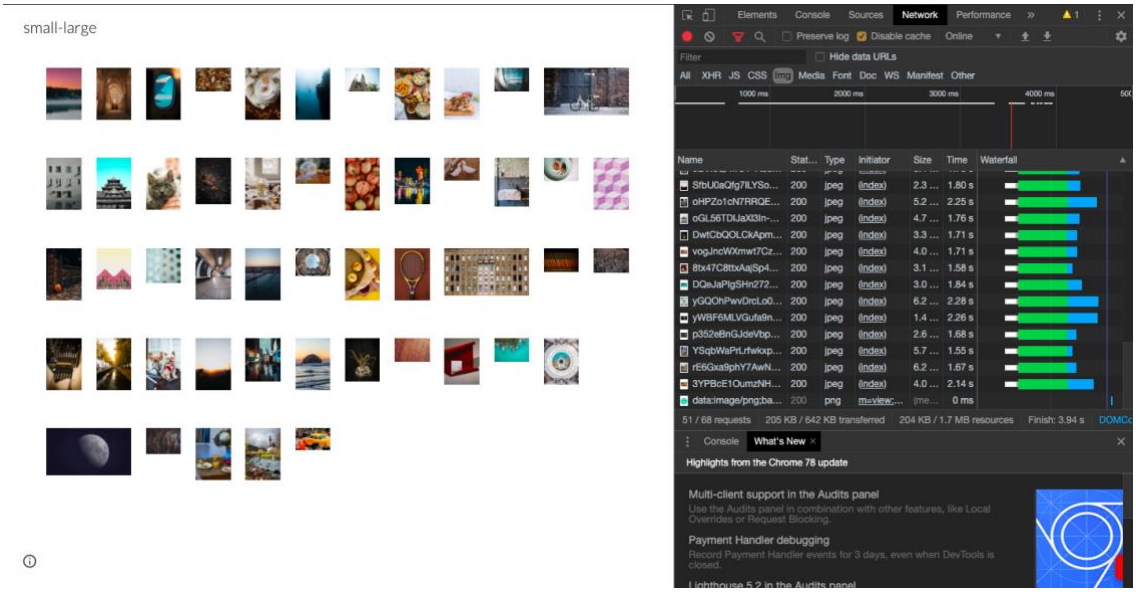
1. Objects which are small – 400 bytes to 8KB
2. Objects which are large – 200KB or above
3. Sites with small number of objects – 5
4. Sites with large number of objects – 50
5. Objects could be of different types pictures, text data, gifs, (paper and this project uses image/jpegs)

Following four google sites were created:

1. Small objects (400 bytes to 8KB), small number of objects (5) (*small-small*),
   Resource size: 1.4 MB
2. Small objects (400 bytes to 8KB), large number of objects (50) (*small-large*),
   Resource size: 1.7 MB
3. Large objects (128KB), small number of objects (5) (*large-small*),
   Resource size: 2.9 MB
4. Large objects (128KB), large number of objects (50) (*large-large*),
   Resource size: 19.6 MB

The images uploaded on these sites are free to use and are randomly sampled from here distributed under a (CC-BY-SA) .

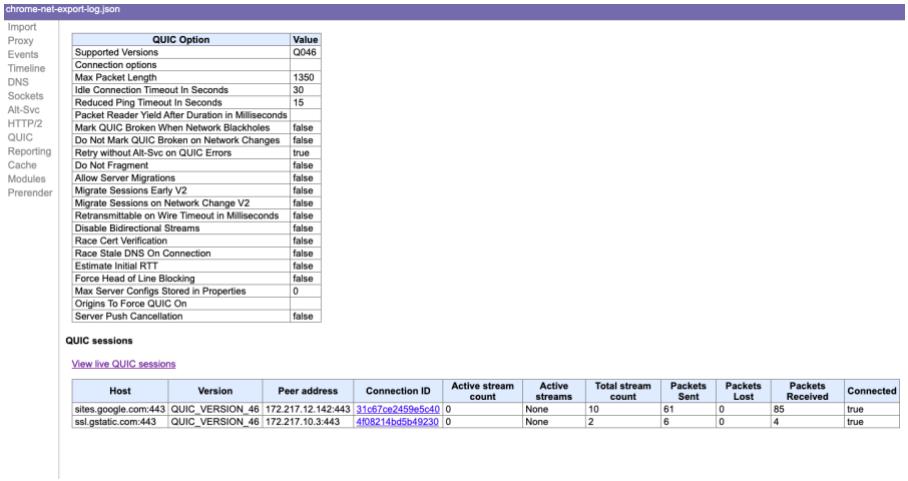
Snapshot of the *small-large* website with 50 images of size less than 8KB and load time with caching disabled is shown below-



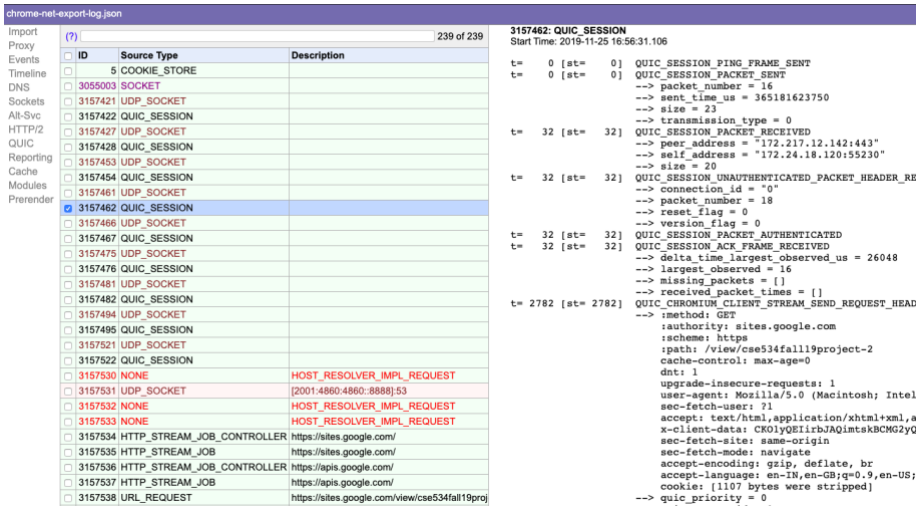
3.4. Enabling Quic

For the headless chrome run, Quic is enabled using the --enable-quic parameter. For the http2 tests, this parameter is not used. By default, Quic is disabled.

For non-headless, GUI case, Quic is enabled by changing the experimental switch in the settings of the chrome. The Quic operation is verified by scrutinizing the log files. Below image shows the example of a Quic version and connection information from the chrome-net-log file.



Example of a **Quic** session in action –



During the experiment run, I verified the usage of Quic as well as several of the protocol's parameter by scanning the HAR file and checking the "alt-svc" key. This is absent when the experiment is run without enabling Quic.


```
{
        "name": "alt-svc",
        "value": "quic=\":443\"; ma=2592000; v=\"46,43\",h3-Q050=\":443\"; ma=2592000,h3-Q049=\":443\";
            ma=2592000,h3-    Q048=\":443\"; ma=2592000,h3-Q046=\":443\"; ma=2592000,h3-Q043=\":443\";
            ma=2592000"
},
```


3.5. Setting up Chrome.

The setup process is given in detail in the *Readme.txt* file. For the headless chrome, ***--enable-benchmarking*** flag is used [12, 13] which

1. Clears the caching between each page load (media and content caches) preventing chrome from loading page contents from the local cache.
2. Clears the existing connections between each page load, tears down and recreates new socket for each connection.
3. The option also enables performance measurements such overall page load time in KB read/written.
4. Could be used in conjugation with parameter "iterations", specifying how many times the test page should be loaded.

3.6. Traffic Shaping

The command line tools ***dnctl*** and ***pfctl*** *[6]* were used to shape the traffic. The steps are covered in *Readme.txt* file. Following network parameters were set up-

1. Small access speed of 2Mbps, large access speed of 10Mbps*.
2. Low loss, the inherent loss of the network and no extra loss added on top. High loss case, a loss of 2% added to the network in upstream and downstream directions.
3. Low delay case, the intrinsic delay of the network, no extra delay added. High delay case – a delay of 100ms was added in upstream and downstream direction with a total delay of 200 ms.

*I could not test for 50 Mbps due to the bandwidth limitation. This was a network parameter tested in the paper. The wifi network is a bit fickle and unreliable, thus requiring a speed test before running an experiment. Moreover, I ran the experiments 50 times each to smooth out noise and manually removed any outliers and discarded results when the PLT generation timed out.

3.7. Parameter sweep:

The following table shows the parameter space and the value sweep-

Category	**Parameter**	**Values**
Network	Bandwidth	2Mbps, 10Mbps
	RTT (sum of up and down)	0ms, 200ms (native latency ~ 15ms)
	Packet loss	0%, 2%
Website objects	Number	5 images, 50 images
	Size	8KB or below, 200KB or above

3.8. Automation

The experiments were automated using the script - automator.py. The script does the following-

1. Runs chrome in headless mode with **Quic** enabled/disabled (disabled for http2)
2. Iterates over each parameter, creates a pipe (filter) using ***dnctl*** and ***pfctl*** and attaches the site to it. Thus shaping the traffic for a particular site.
3. The HAR files are saved and renamed to capture the parameters of the run as well as the website for which they were created. These are saved in /HAR-files.
4. The process is repeated till the parameter sweep is complete.

4. Measurement Results

Cumulative Distribution Function for page load times for above configurations (there are 64 possible PLT plots, 3200 data points, extremes of each cases or cases which are informative/conclusive will be selected.)

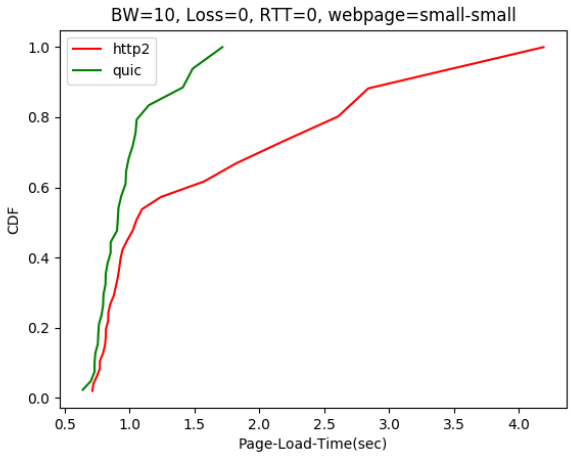
4.1. Measurements

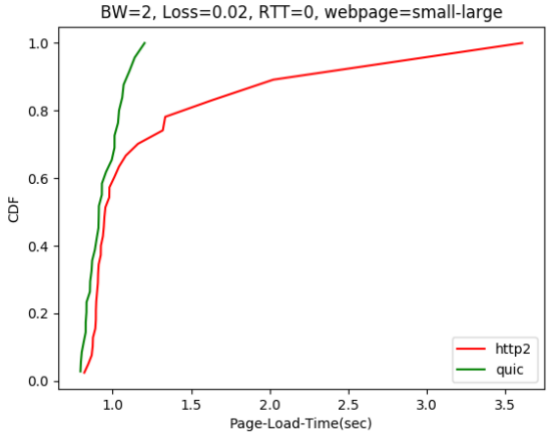
The x-axis in the plot is page load time in seconds, the Y axis is the total fraction of loads for which the page load time is less than or equal to the x-axis value.

4.1.1. When **Quic** and **HTTP2** both are at par.

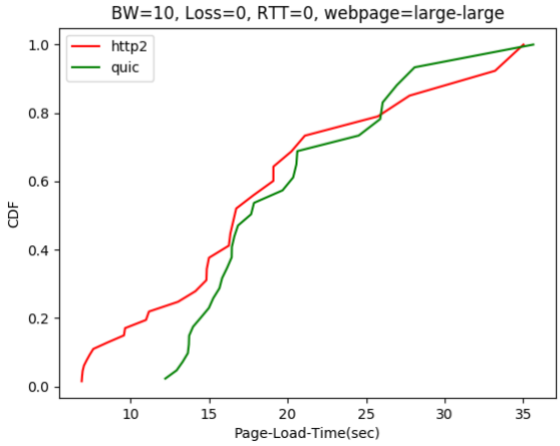
Low loss and low RTT cases (**please note that there is an inherent RTT of about 12-15ms, the RTT shown here is the extra added RTT, 100ms is added to both the uplink and downlink pipe resulting in total delay of 100ms.**)

BW=10Mb/s, Loss=0%, RTT=0ms, small objects, small number





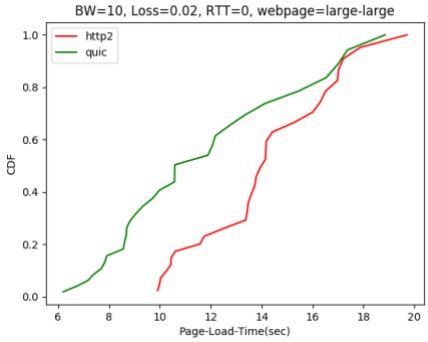
BW=10Mb/s, Loss=0%, RTT=0ms, large objects large number



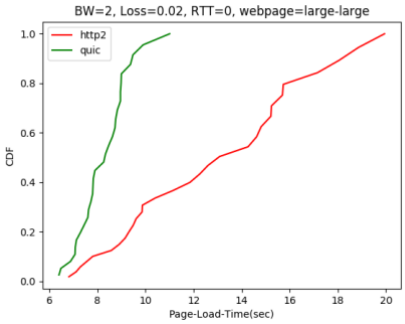
4.1.2. When **Quic** outperforms **HTTP2**

4.1.2.1. High loss cases-

BW=10Mb/s, Loss=2%, RTT=0ms, large objects, large number

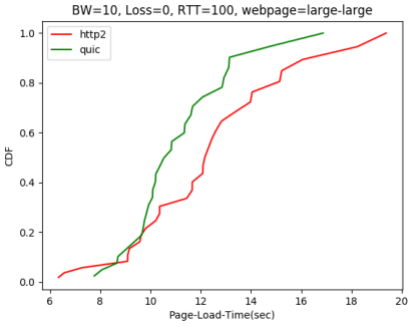


BW=2Mb/s, Loss=2%, RTT=0ms, small objects, large number

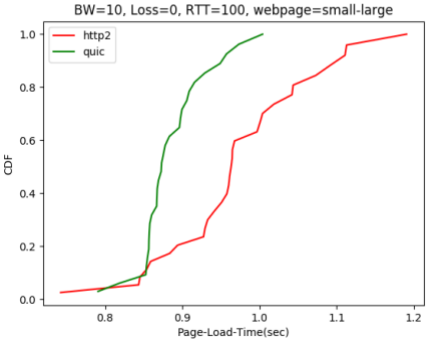


4.1.2.2. High RTT cases –

BW=10Mb/s, Loss=0%, RTT=100ms, large objects, large number



BW=2Mb/s, Loss=0%, RTT=100ms, small objects large number



4.2. Observations

1. **Quic** performs better than HTTP2 under high RTT cases for large number of small images, irrespective of bandwidth or loss.
2. **Quic** performs better than HTTP2 in case high loss, this could be attributed to the forward error correction implementation in Quic.
3. In case of 0 loss and low RTT Quic and HTTP2 are more-or-less equivalent.

5. Conclusions

This project compared Quic and HTTP2, with client and server running on gQuic compatible stack. Network parameters were shaped in three dimensions – loss rate, bandwidth and RTT delay. Following could be concluded from the test results –

1. The performance under high loss scenario suggests that FEC has helped in sustaining page load time and has improved recovery. This significantly benefits Quic in keeping page load time low when compared to traditional HTTP+TCP stack.
2. HTTP2 and Quic were at par while loading small object size websites in low loss cases across bandwidth. This is because both are multiplexed protocols and utilize the transport layer to the maximum.
3. Test Confirms previous finding which states that a high loss environment negatively affects the page load time, but the effect is mitigated in Quic compared to HTTP2.
4. In high RTT case, Quic outperforms HTTP2. With the ability to persist the stream across network provider ans packet pacing, Quic could be advantageous for mobile traffic. The advantage is even more prominent in case of low bandwidth with high RTT.

The project did not cover the very high bandwidth scenario (50Mbps). The paper showed that in this case Quic underperformed compared to HTTP1.1. In the high bandwidth case, the dynamic packet pacing algorithm could be a bottleneck since it could reach its peak in case of very high bandwidth.

References and links:

[1] Wiki page on Quic, https://en.wikipedia.org/wiki/QUIC
[2] "*When to use and when not to use BBR: An empirical analysis and evaluation study*", Yi cao, Arpit Jain, Kriti Sharma et al., Internet Measurement Conference (IMC '19), October 21–23, 2019
[3] "*How Speedy is SPDY?",* Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, NSDI '14
[4] "*How quick is QUIC?*", P´eter Megyesi, Zsolt Kr¨amer, S´andor Moln´ar, IEEE ICC 2016 - Communication QoS, Reliability and Modeling Symposium
[5] "*Does QUIC make the Web faster ?"* Prasenjeet Biswal, Omprakash Gnawali , GLOBECOM, 2016
[6] dnctl and pfctl - https://spin.atomicobject.com/2016/01/05/simulating-poor-network-connectivity-mac-osx/
[7] Tool to capture HAR files in chrome: https://github.com/cyrus-and/chrome-har-capturer
[8] IETF drdaft of http over quic: https://tools.ietf.org/wg/quic/draft-ietf-quic-http/
[9] Various opensource implementation of Quic, https://github.com/quicwg/base-drafts/wiki/Implementations
[10] Nginx server roadmap: https://trac.nginx.org/nginx/milestone/nginx-1.17
[11] Quic usage statistics as of Nov 2019, https://w3techs.com/technologies/details/ce-quic
[12]All chrome switches - https://tinyurl.com/r8kt3fg
[13] Clearing cache and benchmarking - https://www.chromium.org/developers/design-documents/extensions/how-the-extension-system-works/chrome-benchmarking-extension
[14] IETF task force meeting, 14-Nov-2019, https://www.youtube.com/watch?v=FnScP8r9JLg#t=5m26s