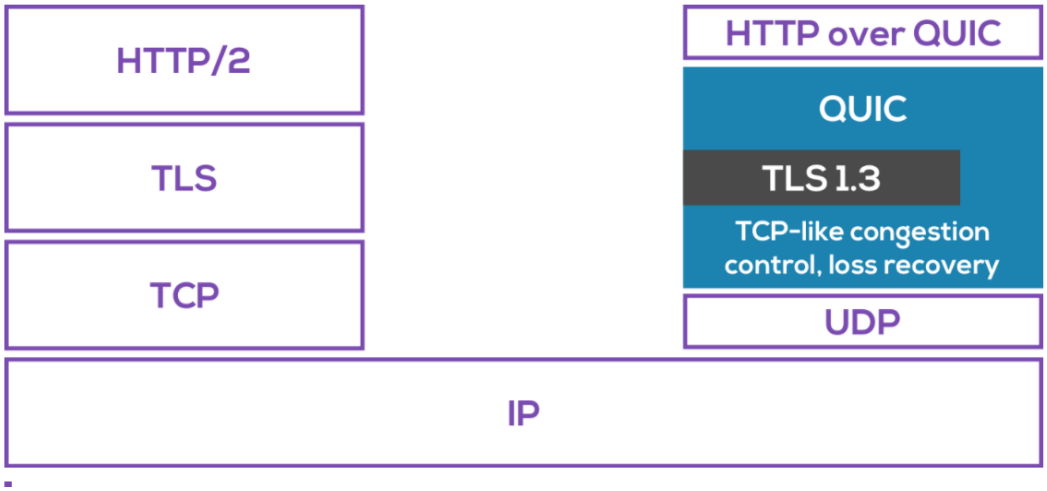
CSE 534 Spring 2017

# QUIC vs HTTP/2 Performance Analysis

*Anusha Muthyampeta (110929473)*
*Parkavi Sundaresan (110946896)*
*Rakesh Agarwal (110945930)*

## INTRODUCTION

Increase in size and complexity of web pages has challenged the efficiency of HTTP.
Recent enhancements made to speed up the web have resulted in QUIC. This project mainly focuses on measuring the performance of websites hosted on HTTP/2 and QUIC protocol in emulated controlled network conditions. Our approach is to sweep the parameter space and evaluate where QUIC currently helps or hurts. This has helped in concluding that though HTTP/2 outperforms QUIC in most scenarios, QUIC offers significant advantage in low bandwidth and high latency links.




## QUIC

QUIC (Quick UDP Internet Connections) is a transport layer network protocol that is built on UDP. On the surface, QUIC is similar to TCP+TLS+HTTP/2 implemented on UDP. Connection establishment requires 0-RTT. It also offers better resilience to packet loss and loss of a flow. Because TCP is implemented in operating system kernels, and middlebox firmware, making significant changes to TCP is next to impossible. However, since QUIC is built on top of UDP, it suffers from no such limitations and hence supports pluggable congestion control. Thus, it outperforms its predecessors over low-bandwidth links and high-RTT links

Key advantages of QUIC over TCP+TLS+HTTP2 include:

- Connection establishment latency
- Improved congestion control
- Multiplexing without head-of-line blocking
- Forward error correction
- Connection migration

## HTTP/2

HTTP/2 is the recent and major revision of the most commonly used HTTP network protocol. The main idea of HTTP/2 is to multiplex the requests over a single TCP connection, instead of the 6 parallel connections as in HTTP/1.1. This reduces the RTT in connection establishment, relatively. In order to work over a single connection, each request /response pair is abstracted as a stream, and streams are multiplexed into packets to be sent over the wire. As a result, several requests can load simultaneously. HTTP2 also features header compression using HPACK algorithm.

Server push is a noticeable feature where server will push the response even before the client requests for them, if it thinks that the client it is highly probable for the clients to request for those objects. For example, it sends all the objects embedded or included in a web page when the html is requested. TCP is implemented in operating system kernel, and hence making significant changes to TCP is difficult. Since HTTP/2 is built on TCP, it is next to impossible to customize features like congestion control algorithm, as required by network conditions. Also, if there is packet loss, whole TCP connection is affected. So, cwnd is reduced. Hence, HTTP/2 does not perform better than QUIC in case of lossy links.

## QUIC SETUP

Fisrt step in QUIC setup is downloading and building the chromium code base on an Ubuntu machine. This involved the following steps:

```
git clone https://chromium.googlesource.com/chromium/tools/depot_tools.git
export PATH="$PATH:/path/to/depot_tools"
mkdir ~/chromium && cd ~/chromium
fetch --nohooks --no-history  chromium
```

After checking out the code, all dependencies were installed and Chromium specific hooks were run. Ninja was used as main build tool along with a tool called GN to generate .ninja files.
And,

```
gn gen out/Default
ninja -C out/Default chrome
out/Default/chrome
```

This took around 21 hours. After this setup, we had built a sample server and client implementation, provided in Chromium.

```
ninja -C out/Debug quic_server quic_client
```

We hosted web pages on Amazon EC2 to be recorded and replayed by the Chromium browser. Below are the commands used to download the web page and load it from the server using quic-client:

- Download the web page and change headers in all the files as required:
  *Mkdir /tmp/quic-data*
  *Cd /tmp/quic-data*
  *wget -p --save-headers https://d28nh1crlzhymi.cloudfront.net/*

- Change headers in the files

  *Remove (if it exists): "Transfer-Encoding: chunked"*

  *Remove (if it exists): "Alternate-Protocol: ..."*

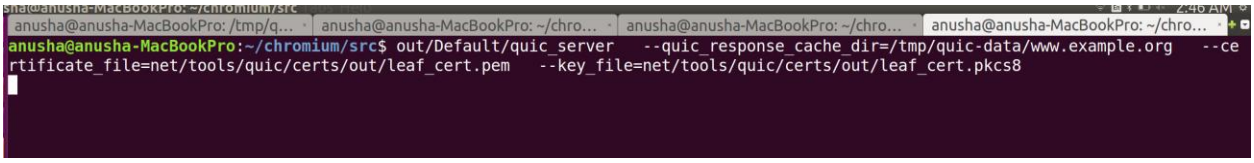
  *Add: X-Original-Url: https://d28nh1crlzhymi.cloudfront.net/*

  *Add: Content-Type: Transfer-Encoding*

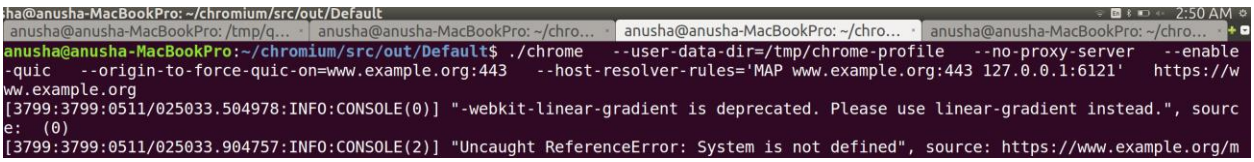
- Generate Certificate for this domain
  *cd net/tools/quic/certs*
  *Change domain in leaf.cnf*
  *./generate-certs.sh*
  *cd out*
  *certutil -d sql:$HOME/.pki/nssdb -A -t "C,," -n <certificate nickname>*
  *-i <certificate filename>*

Now, the server and client were run on two different ports.

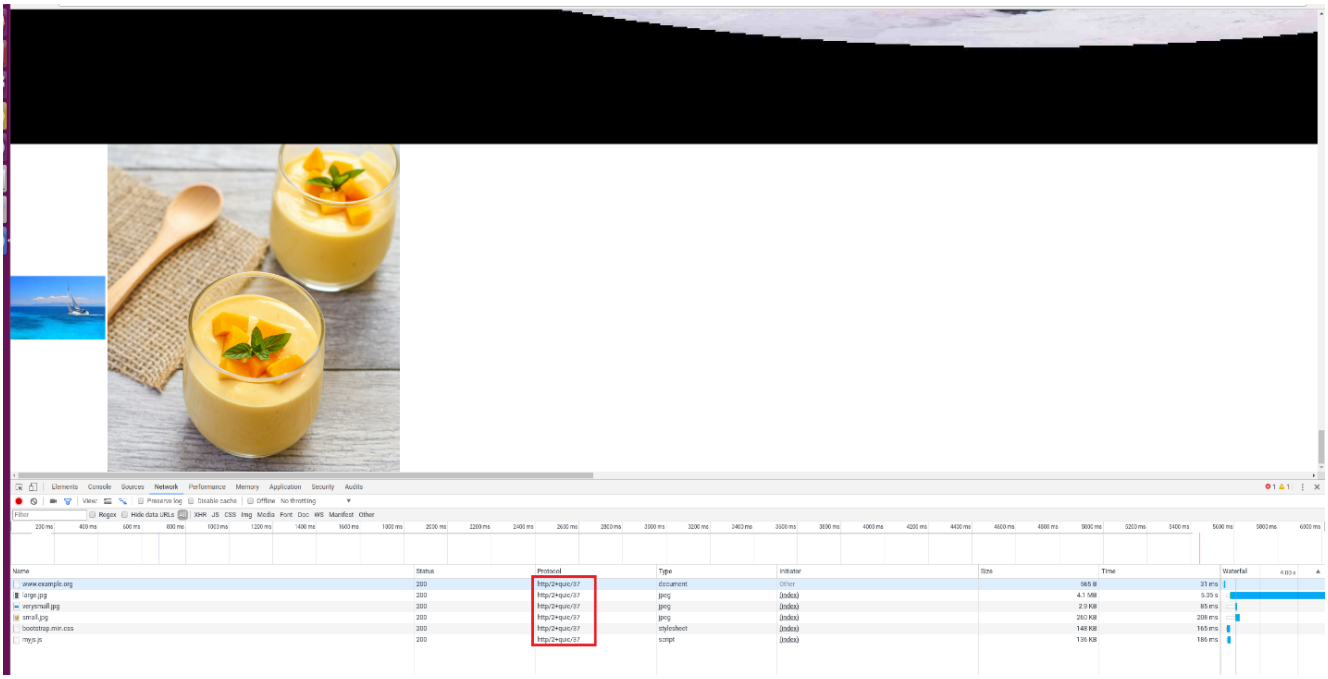
QUIC- Server:




QUIC- Client:



We ensured that the web page is rendered using QUIC protocol through chrome developer tools. A sample web page loaded from the quic server:



## HTTP/2 SETUP

We downloaded latest Apache2 Server and configured it by using *--enable-http2 option.*This enables the module 'http2' which does implement the protocol inside the Apache server. Then, modified httpd.conf to enable http2 module. Also, generated openssl certificate and enabled ssl module. Below are the lines added to httpd.conf:

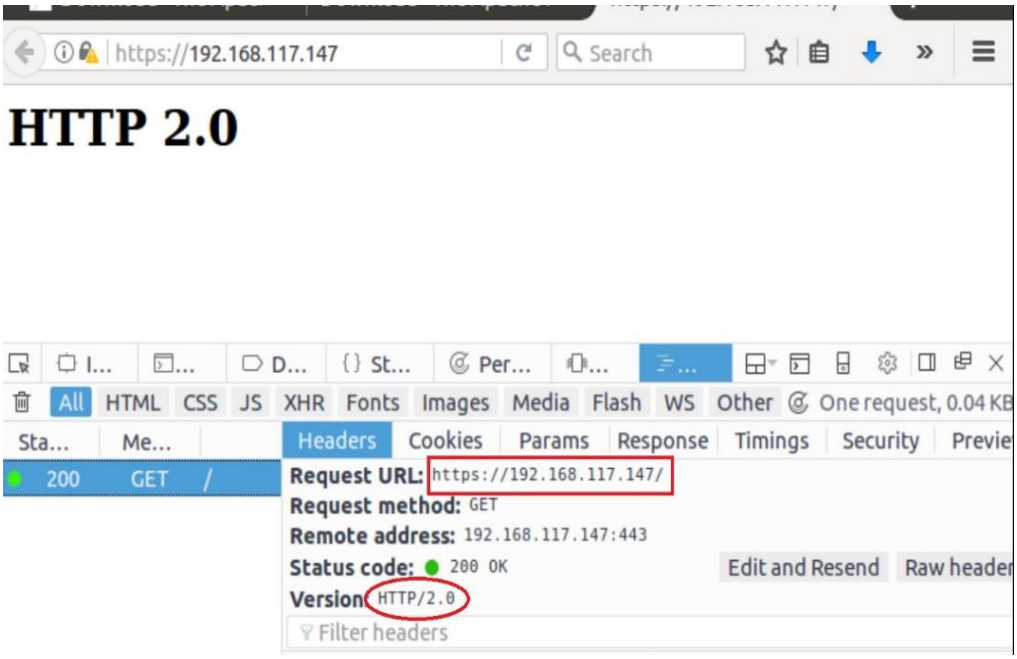
*Listen 443*
*<VirtualHost *:443>*
*SSLEngine on*
*SSLCertificateFile "/usr/local/apache2/certs/test.crt"*
*SSLCertificateKeyFile "/usr/local/apache2/certs/test.key"*
*</VirtualHost>*

*SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-DSS-AES128-GCM-SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-*

*SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES256-SHA256:DHE-DSS-AES256-SHA:DHE-RSA-AES256-SHA:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!3DES:!MD5:!PSK*

*SSLProtocol All -SSLv2 -SSLv3*

Then, hosted the same web pages on the server as QUIC and recorded Page Load Time for different parameters based on objects as well as network conditions.



# PERFORMANCE ANALYSIS

In order to compare the performance of QUIC over HTTP/2, we measured Page Load Time by varying different parameters – objects in the web page as well as network conditions. Then, recorded values were plotted using d3.js. We used multi-scale bar chart and line chart to visualize QUIC and HTTP2 performance for different criteria.

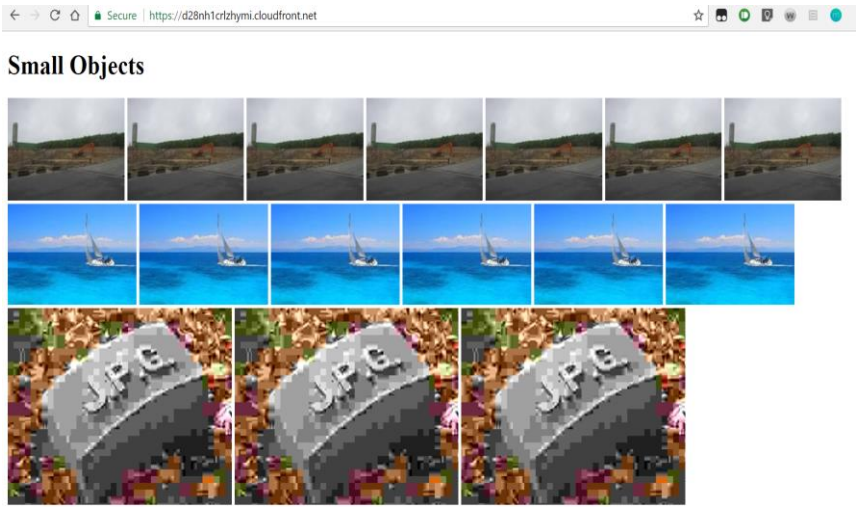
The first three set of metrics were obtained by varying the number of objects on the web page loaded –

- Number of Objects
- Type of Objects
- Size of Objects

These were evaluated under the best network conditions – High bandwidth and Low latency.

## Number of Objects:

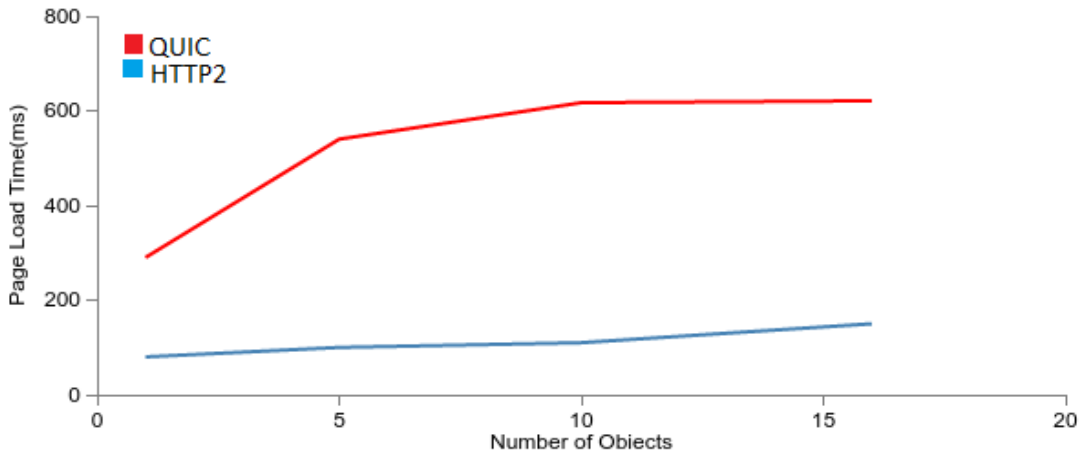
Small Images up to 3KB were used to get this metrics. PLT was noted by increasing the number of objects on the web page.




| No of Objects | QUIC(ms) | HTTP/2(ms) |
|---|---|---|
| 1 | 290 | 80 |
| 5 | 540 | 100 |
| 10 | 617 | 110 |

| | | |
|---|---|---|
| 16 | 621 | 150 |

It was observed that HTTP/2 loads faster than QUIC but as the number of objects increases, difference in performance decreases.

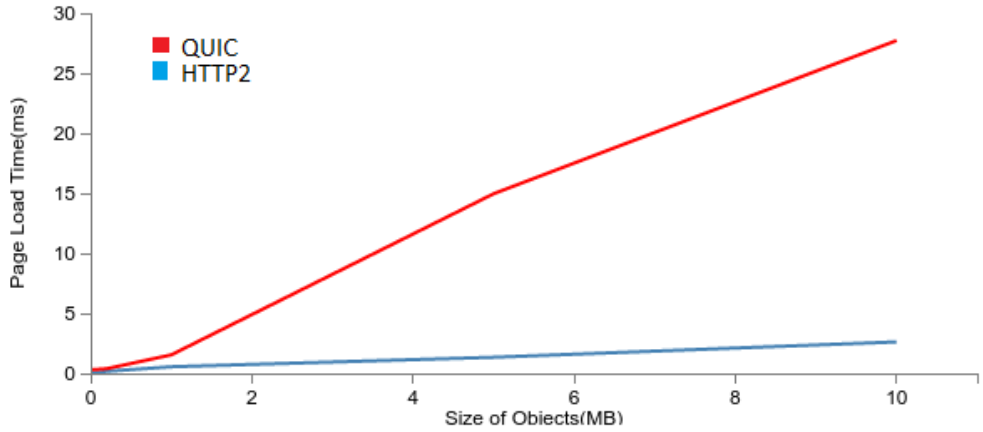



## **Size of Objects**:

Size of the object on the web page loaded was increased to study the change in Page Load Time. Images of varying sizes were loaded.

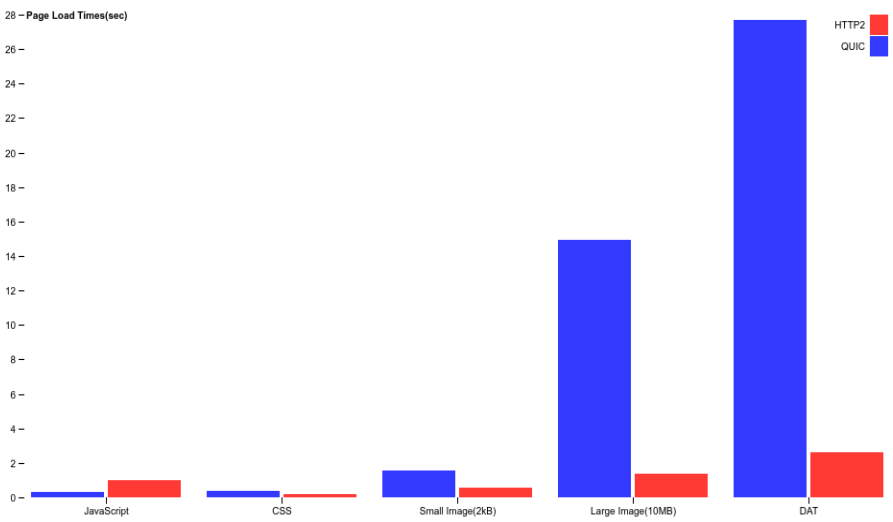
| Size of Objects | QUIC | HTTP/2 |
|---|---|---|
| 1KB | 290ms | 10ms |
| 200 KB | 388ms | 170ms |
| 1MB | 1.54s | 0.55s |
| 5MB | 14.96s | 1.34s |
| 10MB | 27.72s | 2.62s |

PLT was again more for QUIC when compared to HTTP/2 but QUIC seemed to perform better when there were a number of large files to be loaded.




**Type of Objects:**

| Type of Object | QUIC | HTTP/2 |
|---|---|---|
| js | 385ms | 30ms |
| css | 430ms | 30ms |
| small image | 290ms | 10ms |
| large image | 27.72s | 2.62s |
| Dat file | 195ms | 80ms |




## Under Different Network Conditions:

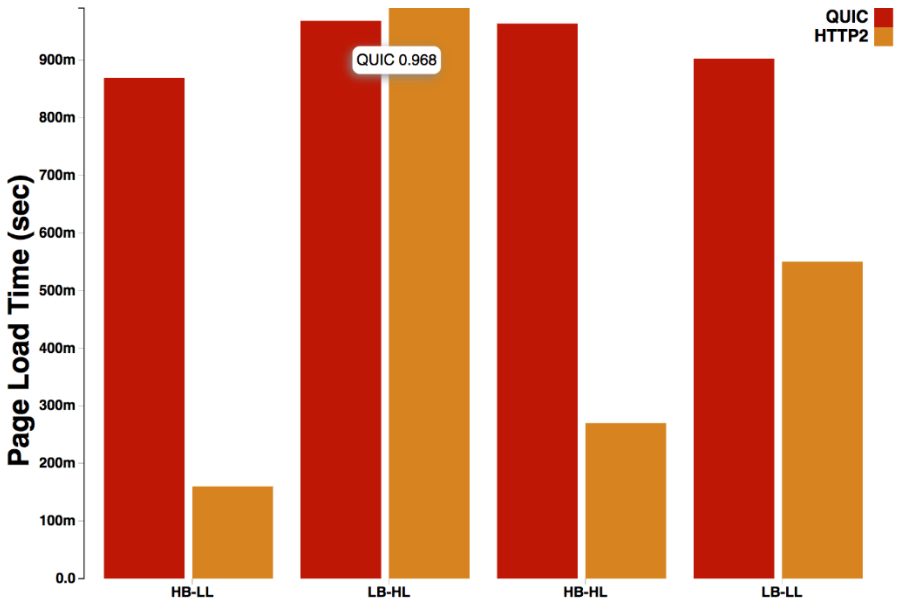
We tested how QUIC and HTTP/2 perform under different network conditions by creating 4 custom conditions in the Network Setup in Developer Tools, on the Chromium Browsers. Three web pages – with number of small images, with few large images and with different type of objects – were created and tested on both QUIC and HTTP/2, under each of the four conditions below. First value denotes the latency. Second and third are the download and upload link bandwidths.

- High Bandwidth Low Latency – 2ms, 30Mb/s, 15Mb/s
- Low Bandwidth High Latency – 100ms, 750kb/s, 250kb/s
- High Bandwidth High Latency – 100ms, 30Mb/s,15Mb/s
- Low Bandwidth Low Latency – 2ms, 750kb/s, 250kb/s

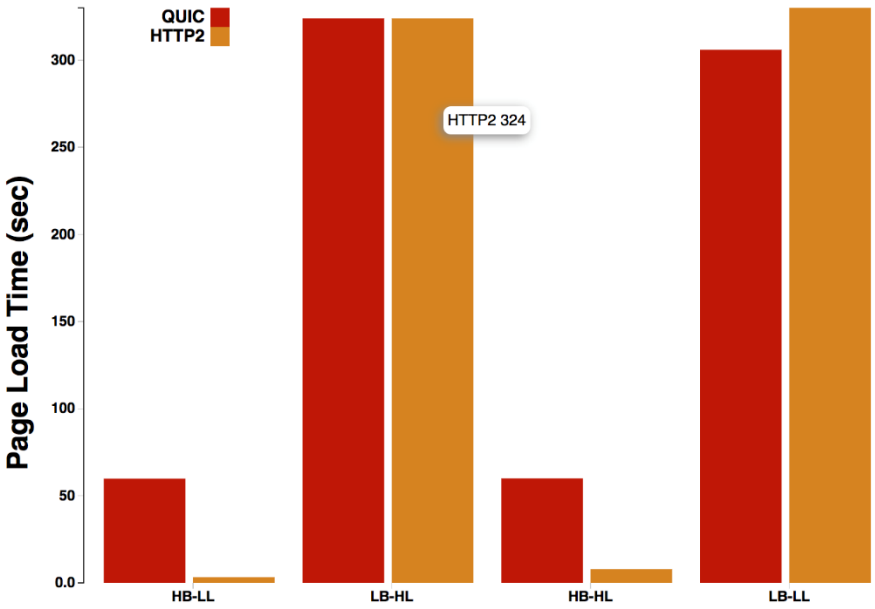
Below is the table with the ***PLT ( in seconds )*** of the mentioned three web pages under different network conditions.

| Web page | HB- LL | | LB- HL | | HB - HL | | LB - LL | |
|---|---|---|---|---|---|---|---|---|
| | QUIC | HTTP2 | QUIC | HTTP2 | QUIC | HTTP2 | QUIC | HTTP2 |
| Many small objects | 0.869 | 0.16 | 0.968 | 0.99 | 0.963 | 0.27 | 0.902 | 0.55 |
| Few large objects | 59.84 | 3.25 | 318 | 324 | 60 | 7.94 | 306 | 330 |
| Different objects | 9.17 | 0.81 | 60 | 53.99 | 8.49 | 1.42 | 50.18s | 51.50 |

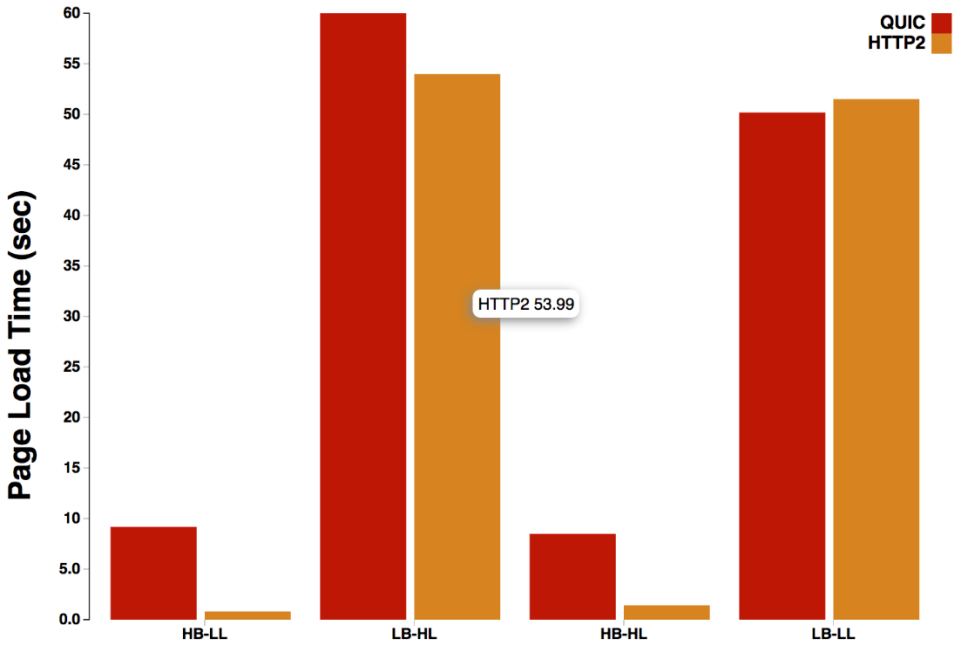
**Web page with a number of Small Objects :**



## Web page with a few Large Objects:




## Web page with different type of Objects:



## INFERENCES & LIMITATIONS:

- From the plot of PLT against number of objects, we can see that although HTTP/2 outperforms QUIC, the difference tends to decrease as the number of objects increase.
- From the plot of PLT against size of objects, we can see that HTTP/2 outperforms QUIC, the difference tends to increase with size
- From the data under varying network conditions, we can observe that QUIC has a better or comparable performance w.r.t HTTP/2 in low bandwidth conditions and high latency conditions.
- While loading a web page with a large object and then small objects, we could observe that there was no Head-of-Line Blocking in case of QUIC
- We used an optimized module in Apache2 to enable HTTP/2 but a toy server for QUIC. Due to this server setup inconsistencies between QUIC and HTTP/2 we cannot be certain about the validity of PLT comparisons. But, we do observe a trend that the difference in performance decreases drastically in low bandwidth conditions.

## CONCLUSION:

Based on our experiments we can conclusively say that QUIC outperforms HTTP/2 in Low-Bandwidth conditions. This can be attributed to its header compression feature and low start overhead. The performance difference between QUIC and HTTP/2 decreases in case of high latency. This is because of 0-RTT connections in QUIC.

Although we see that QUIC performs better than HTTP/2 in certain conditions, it is still not a ubiquitously better protocol. Thus, we need more extensive studies under much consistent and controlled conditions to provide conclusive claims.

## LINK TO GITHUB REPOSITORY:

*https://github.com/parkavi2609/QUICvsHTTP2*

## BIBLIOGRAPHY:


***QUIC Setup***: *https://www.chromium.org/quic/playing-with-quic*
***HTTP Setup***: *https://icing.github.io/mod_h2/howto.html#https*
*Evaluation of QUIC on Web Page Performance by Somak R. Das*
*http://bl.ocks.org/juan-cb/ac731adaeadd3e855d26*
*Does QUIC make the Web faster ?*
*https://www.chromium.org/quic*
*https://www.reddit.com/r/devops/comments/4st8fc/http_2_vs_http_11_vs_spdy_vs_quic/*