QUIC vs HTTPS
Project Report

Group Members: Kriti Sharma (112101132), Nikita Luthra (112073741)

Introduction:

QUIC (Quick UDP Internet Connections), is a new transport layer protocol developed by Google. Despite being a transport layer protocol, this is deployed at the application layer. The basic idea behind doing this was to be able accommodate new changes and test different algorithms for the transport layer, which was not possible while using TCP.
A brief overview of QUIC would include it’s 5 major advantages over the HTTP2+TLS+TCP combination:

1) Connection Establishment
2) Congestion control
3) Multiplexing
4) Forward Error Correction
5) Connection Establishment

1. Connection Establishment

For a TCP connection, it requires a 3-way handshake and 2 round trips were required to setup TLS. QUIC improves this latency by providing a 1-RTT handshake for setup and subsequently allowing 0-RTT connections.

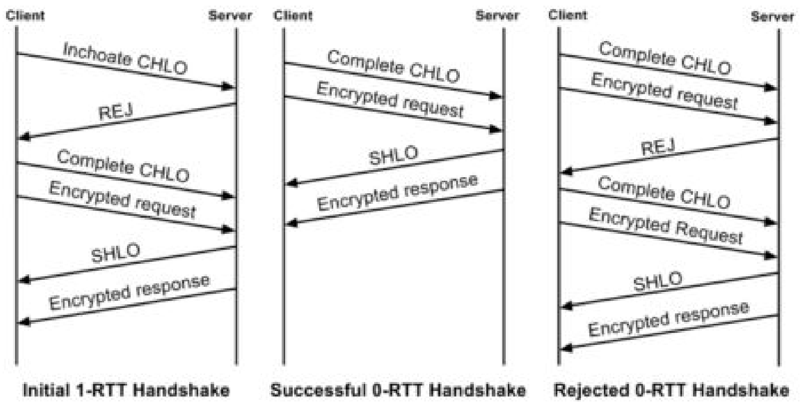


Figure 4: Timeline of QUIC’s initial 1-RTT handshake, a subsequent successful 0-RTT handshake, and a failed 0-RTT handshake.

For the 1-RTT handshake, client sends a “Client Hello” message and the server responds with a “Rejection” message and provides the session key, unique connection ID and other required parameters for the setup. Client caches and stores the value of these parameters.
For the 0-RTT connection, the client uses the cached data to send encrypted messages and requests to the server.

2. Congestion control

QUIC uses new sequence numbers for it's retransmitted packets. This helps in avoiding the retransmission ambiguity. Retransmission ambiguity occurs when the sender fails to recognise whether the received ACK was for the retransmitted or the original packet.

In QUIC, the data is transmitted over streams and the delivery order is determined by stream offsets encoded within the stream frames.

Also, along with having the monotonically increasing sequence numbers, ACKs have a delay field. This delay is the difference in the time when a packet was received at the receiver's end and when the acknowledgement was sent. These properties help in calculating the RTT more precisely.

3. Multiplexing

At the application layer, TCP is seen as a stream of bytes. Any methods incorporated by HTTP/2 to remove the problem of head-of-line blocking, become insignificant at the transport layer as it just sends the packet present in it's buffer. Thus, when a TCP packet is lost, no other data can be sent until the packet is retransmitted and received at the farther end.

Whereas in QUIC, the data is transmitted on various streams. So, a packet lost on one stream does not impact the other streams.

4. Forward Error Correction

QUIC uses forward error correction (FEC) in order to reduce the number of retransmissions. A group of packets is padded with an extra FEC packet. This packet contains parity of all the data packets in the FEC group. If any of the data packets is lost, it can be recovered by using the FEC packet and the remaining packets.

5. Connection Migration

During the setup, every client receives a unique 64-bit connection ID from the server which is randomly generated at the server end. This ID is used by the client for subsequent communications with the server. This connection ID allows connections to survive changes in the client's IP or port address. This allows the client to communicate with the server using its old ID even after shifting from a WiFi to cellular network. This consistent ID can be used to allow migration of the connection to a new server IP address also. It provides an automatic cryptographic verification of the client.

Problem Statement:

Our intention with this project is to analyze and compare Google's application level - transport layer protocol with the existing HTTPS - TCP combination. We compared both these protocols based on Page Load Times of websites using these protocols and found interesting results.

Approach and Solution:

The Quic Project is part of Open Source - Google Chromium project. It is well maintained and supported for Ubuntu. Therefore, we decided to develop our project on Ubuntu 14.04, Trusty Tahr. The project is very heavy in size and requires about 100 GB of free disc space on the machine it is being developed on.

To get started with the project, the first task that we identified was to be able to run Quic Server and Client on our local machine. To do this we followed the Local Quic Client and Server link's instructions. Going by the instructions we first downloaded the most recent Quic code and installed all the dependencies. After this, we built the binaries- Quic_server, Quic_Client and the chromium browser. A special build system called "ninja" was used to complete this task. Ninja is designed for speed. It is used to build very heavy binaries, in less amount of time.

To provide encrypted transmission, SSL certification was required. To do that, we generated ssl certificate, key and a CA certificate. This CA certificate was added in the Ubuntu's root certificate store. This was done, so that the this certificate is known to web browser like Google Chrome.

After that, we created a cache directory. Here we saved all the index.html (using "wget" command) of the websites which we wanted to serve via Quic server. For each website we added "X-Original-Url" header and removed "Transfer-Encoding: chunked" header.
After this we tested the Quic server and client by running them through command line interface. Quic server took three arguments namely - the path to cache directory, the path to server certificate and key. Like the server, the Quic Client also took three arguments - localhost ip, port number and the url of the website to be requested. Using this we were able to get the html file for the requested website on command-line-interface.

The next task was to build an HTTPS server that provided the same functionality. We developed the HTTPS server using Flask 1.0.2 and Python 2.7. For secure transmission in this case too, we generated self - signed ssl certificates (using openssl). In this case too, we added the CA certificate in the ubuntu's root certificate store.
To serve the same files via both the servers, we copied index.html of the website to be served via HTTPS server from Quic_Cahe to HTTPS_Cache directory. Once everything was in place, we started the server and we were able to load the website in chrome browser locally.

Now that both servers were serving the websites, we wanted to capture their web transactions. We decided to do this using HAR files. One of the easiest way to get HAR file is via Google Chrome. Now, we could easily query the HTTPS server via Google Chrome. However, to query the local Quic server, we had to start it using some special flags.

These special flags were
--enable-quic - to enable quic on the browser.
--origin-to-force-quic-on=www.iana.org:443 - to notify browser to use quic when this url is requested.
--host-resolver-rules='MAP www.iana.org:443 127.0.0.1:6121' - to tell browser to redirect the request to local Quic server ip when a url, "www.iana.org" in this case, is requested.
Using these special flags, we were able to contact the Quic_server using Google-Chrome.

With all the system in place we were all set to evaluate Quic against HTTPS server.

Evaluation setup:

At the start, we selected a total of 21 websites to evaluate the performance of QUIC and HTTPS server. These websites included 19 of the top 25 websites claimed by Alexa.com along with "iana.org" and "example.com". These websites were not of a single type or domain and varied in terms of their content-type and size. The page load times (PLTs) for each website was calculated at least 10 times for each server.

As explained above, both the QUIC and HTTPS server were running on the same machine. We used Google Chrome as a client to send requests to both of these servers, but the procedure to use this client was different for QUIC and HTTPS. This is explained in the procedure below.

For each website, the procedure was as follows:

- ➔ Start the QUIC server from the command line by giving information about the folder containing the html page of that website and the certificate folder.
- ➔ Start the HTTPS server by giving information about the folder containing the html page of the website.
- ➔ For the QUIC client, as explained in the earlier section, we needed to start a different instance of Google Chrome with some special flags to enable quic and set up the host resolver rules so that the website url when accessed is done so using our QUIC server.
- ➔ After starting this instance of Google Chrome, we could view our website and capture the page load times using the HAR files. This was done at least 10 times for each website.
- ➔ Next, to capture the page load times using the HTTPS server, we directly accessed the website on our HTTPS server using a standard Google Chrome browser with the following url: *"https://localhost:5000"*; where 5000 was the port of our HTTPS server.
- ➔ After loading the website, we captured the page load times in a way similar to that of QUIC, by using the HAR files. This was done at least 10 times for each website.

After noting down the load times for each website, we calculated the average PLT for each website and each server and compared these values to get the results as explained below.

Results:

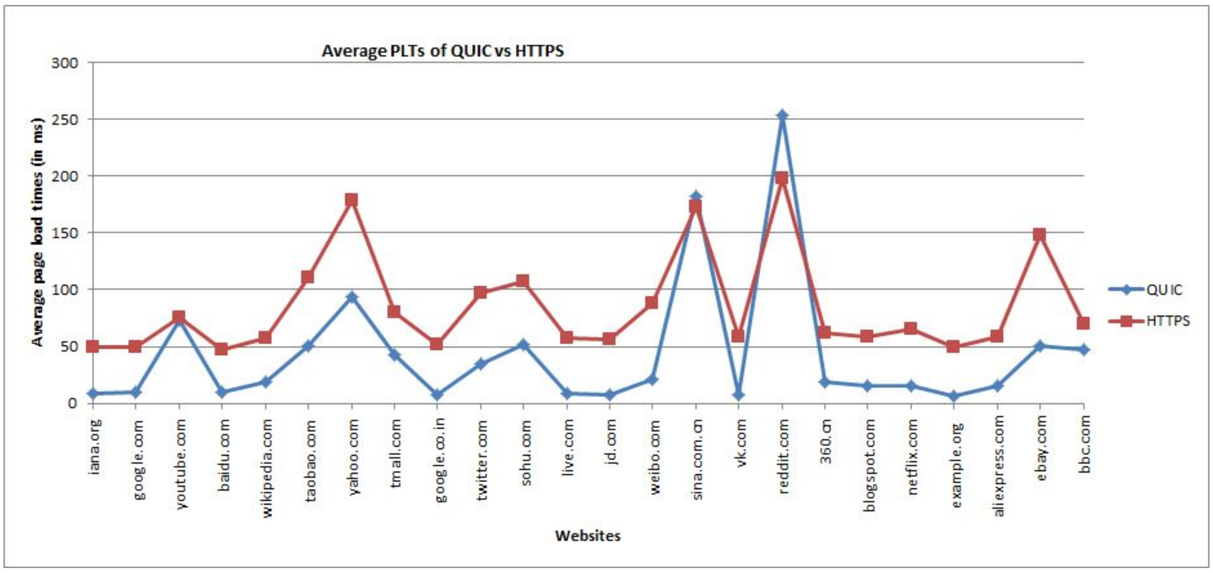
The page load times of the websites selected at the start of the evaluation are as follows:

Name of the website	Avg Quic Page Load Time (ms)	Avg Https Page Load Time(ms)
www.iana.org	9	50
www.google.com	9.3	49.7
www.youtube.com	73.7	75.3
www.baidu.com	9.2	46.9
www.wikipedia.com	18.5	57.7
www.taobao.com	50.5	110.3
www.yahoo.com	94.12	178.87
www.tmall.com	43.09	80.45
www.google.co.in	7.9	51.36
www.twitter.com	35.08	97.58
www.sohu.com	51.72	107.81
www.live.com	8.58	57.16
www.jd.com	7	56.16
www.weibo.com	21.54	87.54
www.sina.com.cn	182.6	172.6
www.vk.com	7.69	58.53
www.reddit.com	253.27	198.54
www.360.cn	18.6	61.4
www.blogspot.com	15.9	58

www.netflix.com	15	64.8
www.example.org	6.9	49.1

For majority of the websites, the QUIC server was much faster than HTTPS with more than **50% improvement** in page load times. But we saw a few anomalies (indicated by the red border), from the list of 21 websites, we found some websites like “reddit.com”, “youtube.com” and “sina.com.cn” in which the QUIC performance was comparable or even worse than HTTPS.

This can also be seen from the graph:



Intuitively we thought this to be because these websites have a large number of small objects. To explore this phenomenon further, we checked the PLTs for websites having this property.

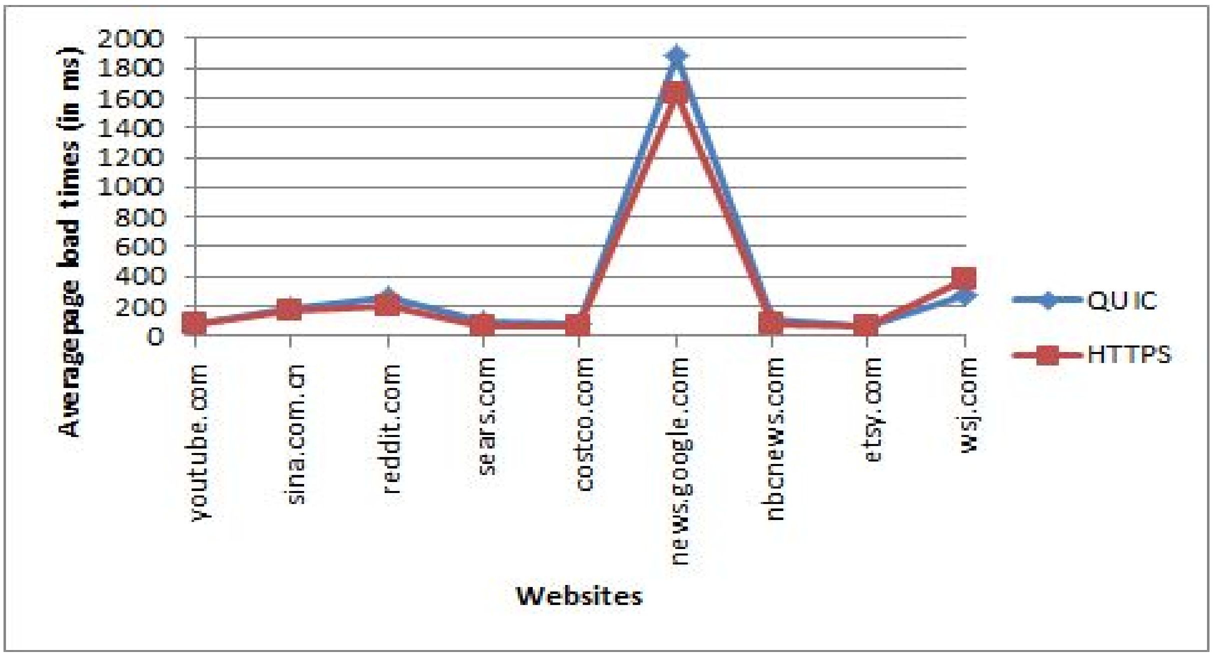
Additional evaluations:

We found different websites having a large number of small objects. This was done by checking the number and size of the objects in the website using an online evaluation website “webpagetest.org”. These new websites mostly included news websites.
The page load times were as follows:

Name of the website	Avg Quic Page Load Time (ms)	Avg Https Page Load Time(ms)
www.aliexpress.com	15.2	58.1
www.ebay.com	50.7	148.3

www.bbc.com	47	70.3
www.sears.com	95.6	70.2
www.costco.com	78.9	58.6
www.news.google.com	1890	1623
www.nbcnews.com	108.7	73.3
www.etsy.com	61.3	67.1
www.wsj.com	273.6	376.6

In these set of websites, there were still some, where QUIC performed better, but for most of the websites QUIC was comparable and at times even worse than HTTPS.



Contradicting to what we expected, the “google news” website also had a large page load time using QUIC server as compared to HTTPS server.

Further, after discussion with the professor, we wanted to evaluate whether these anomalies was because the client and server were on the same machine and had no delay in getting a response because of the network. So, we introduced a 30ms delay in the network and calculated the page load times again. Interestingly, the results were actually a bit different for the websites having large number of small objects.

To add a delay we used the “tc” command that added a delay of 30ms to all the localhost traffic.

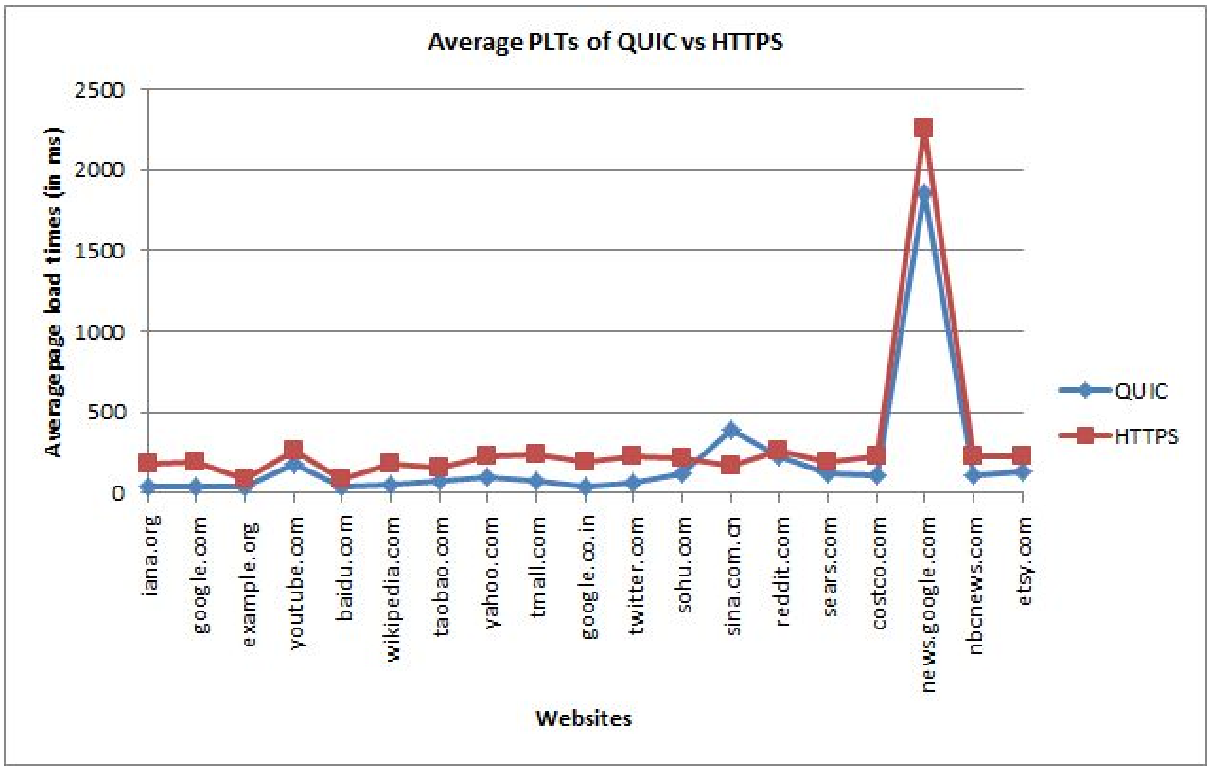
```
sudo ifconfig lo:1 127.0.10.10 netmask 255.0.0.0 up
sudo tc qdisc add dev lo:1 root netem delay 15ms
```

The page load times are as follows:

Name of the website	Avg Quic Page Load Time (ms)	Avg Https Page Load Time(ms)
www.iana.org	38.3	181.6
www.google.com	40.6	187.5
www.example.org	37.9	80.4
www.youtube.com	179.3	259.9
www.baidu.com	37.2	82.8
www.wikipedia.com	50.7	185.4
www.taobao.com	76.273	152.82
www.yahoo.com	95.1875	225.4375
www.tmall.com	74.272	235.455
www.google.co.in	36.545	187.091
www.twitter.com	60.083	225.083
www.sohu.com	117.73	210.64
www.sina.com.cn	395.3	168.9
www.reddit.com	224.81	256.19
www.sears.com	115.8	195.2
www.costco.com	112	226.7
www.news.google.com	1853	2253
www.nbcnews.com	110.2	231.3
www.etsy.com	129.8	226.2

For almost all websites even including the ones having large number of small objects , QUIC performed better than HTTPS and gave more than **50% improvement** in page load times.

But, there were 2 websites “sina.com.cn” and “reddit.com”, where QUIC was worse in the former and comparable to HTTPS in the later.
This can be seen in the graph:



Conclusions:

By analyzing the protocol Quic with HTTPS we can say:

1. The Average PLT for majority of websites using Quic is much smaller as compared with the HTTPS-TCP combination and gives an improvement of around 50% for most of them.

2. However, websites which request a large number of objects along with index.html display a different behaviour and Quic performs poorly while loading these websites. But, this was found to be true only when there is a small delay in the network between the server and client.

3. When tested by introducing a network delay, we found that QUIC was significantly faster than HTTPS for almost all the websites, but still had a few outliers.

Github Repository Link: https://github.com/kritisharma492/Quic-Vs-Https

References:

1. Arash Molavi Kakhki , Samuel Jero , David Choffnes , Cristina Nita-Rotaru , Alan Mislove, Taking a long look at QUIC: an approach for rigorous evaluation of rapidly evolving transport protocols, Proceedings of the 2017 Internet Measurement Conference, November 01-03, 2017, London, United Kingdom [doi>10.1145/3131365.3131368]

2. Alexa top websites: https://www.alexa.com/topsites
3. Website to evaluate number of objects and page size: https://www.webpagetest.org/

4. QUIC Internet draft: https://tools.ietf.org/html/draft-ietf-quic-transport-08#page-8

5. Taking a long look at QUIC github: https://arashmolavi.github.io/quic/

6. The Chromium Projects: http://www.chromium.org/developers/how-tos/get-the-code

7. QUIC toy server and client setup: https://www.chromium.org/quic/playing-with-quic

8. Linux Certificate Management: https://chromium.googlesource.com/chromium/src/+/master/docs/linux_cert_management.md