Visualization tool to compare HTTP/1.1, SPDY, QUIC

Pushkar Garg
pushkar.garg@stonybrook.edu

Shubham Singhal
shubham.singhal@stonybrook.edu

Shweta Srivastava
shweta.srivastava@stonybrook.edu

Code Repository URL:
https://github.com/pushkargarg/compare-protocols

Visualization Tool URL:
http://allv28.all.cs.stonybrook.edu/shsinghal/FCN/

1. Introduction

In today's internet, the tolerance threshold for how long a user will wait for a slow loading website continues to decrease. Users demand near real time responses which is not possible with HTTP/1.1. One of the drawbacks of HTTP/1.1 is opening of too many TCP connections to achieve concurrency, as, a large number of connections may lead to network congestion and hence to an even worse performance. Second disadvantage is that, the server cannot initiate a transfer and hence response time in case of embedded objects is significantly high. SPDY rectified above shortcomings of HTTP/1.1 by introducing server push, request prioritization and combining of multiple small http requests and response into one packet. It uses multiplexing over single TCP connection and hence avoids congestion. Server push allows server to send data before explicit request from client improving latency in case of embedded objects. It also provides header compression resulting in fewer bytes that need to be transferred. One issue that arises with multiplexing in SPDY is that since it is done over single TCP connection that needs to provide ordered delivery, if one packet is lost all subsequent packets in all streams will have to wait for its retransmission (Head of Line Blocking). QUIC has been developed over UDP and retransmission and congestion control modules for it have been developed at application layer. Since UDP does not enforce in-order delivery, QUIC provides multiplexing with no head of line blocking arising in case of packet loss. Also, in case a connection was already established between client and server in recent past, QUIC uses Connection ID (CID) to identify connections instead of IP addresses which might change on change with network interface, resulting into 0-RTT re-connection cost. The ACK frames in QUIC also contain the delay between the time packet was received and its acknowledgement was sent resulting into estimation of precise RTT. But, the 0-RTT re-connection cost is applicable only if same server is hit otherwise new connection would need to be established.

2. Problem Description

The goal of this project is to create a tool to visualize and differentiate between the flow of packets for HTTP, SPDY and QUIC and to investigate the effectiveness of SPDY and QUIC over HTTP by comparing their performances on general web traffic and heavy latency websites such as YouTube. The tool will allow comparison of any two or all three protocols at a time. The contribution of this project is twofold: i) We provide a tool to visualize differences in handshake and flow of the three protocols. ii) We asses and provide a tool to visualize the performances of the three protocols in terms of average RTT per object and Handshake cost.

3. Measurement Environment

We used a regular laptop with Chrome and Firefox browsers and Wireshark to capture network packets as pcap files. We, then developed packet parsers for all the three protocols in order to parse the pcap files so as to obtain the flow information and page load time and handshake cost values for comparison. Packets were captured for all three protocols on general web traffic as well as streaming data traffic. One limitation we faced here is that since QUIC has been deployed only on Google and YouTube there wasn't much option available to capture QUIC packets. The websites for which data was captured were: google.com (flow while loading home page), gmail.com (flow while loading sign in page), google translate (flow while translating a sample German text), google image search (flow when icon size logos are searched), youTube.com (flow during full streaming of a 50 seconds long video), spdy.centminmod.com/flags.html, facebook.com (flow while loading home page of a signed in user), linkedin.com (flow while loading home page of a signed in user), twitter.com(flow while loading sign in page), en.wikipedia.com(flow while loading home page).

4. Solution Methodology

The module has been developed using JAVA language and JnetPcap library. For visualization tool, D3 has been used. Pcap packets were captured through Wireshark. Pcap files are then given as input to QUIC Parser and SPDY/HTTP 1.1 Parser which output the CSV files containing information such as source and destination IP address, port number, sequence and acknowledgement numbers, TLS information (in case of SPDY) and server configurations, client hello and server rejection (in case of QUIC) and performance matrix (page load time). The visualization tool uses these csv files as input in order to display data. It has a drop down menu from which a user can select whether he wants to compare protocols based on flow (handshake and reconnection) or based on page load time.

Figure 1 shows a snapshot of the visualization tool. On upper left hand corner, a drop-down menu provides users with options such as to compare protocols based on Handshake flow, Reconnection and Page load Time. Another drop down menu provides them to compare data for different websites, such as, google.com, gmail.com, facebook.com etc.

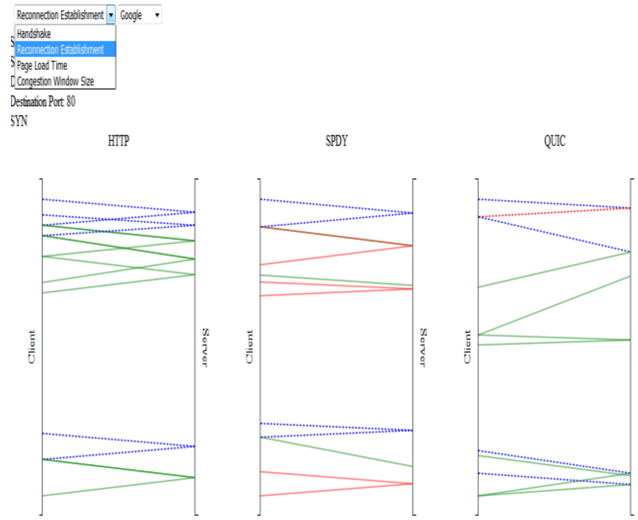


Figure 1: Visualization Tool

5. Evaluation And Results

5.1 Handshake and Reconnection in QUIC:

The initial handshake in QUIC protocol requires 1 RTT. When a client tries to connect with a server for the first time, it sends a Client Hello message which has a SNI (Server Name Indication) tag but not a STK (Source address token) or SCID (Server Config ID) [Figure 1.1] which the server requires in order to verify client's authentication. The server then responds with a connection rejection message sending along its configurations SCFG (Server Config) [Figure 1.2] that contains its configuration ID, information about authentication encryption algorithm, etc. and is valid in general for a few days. The client then re-sends a Client Hello with valid configurations and an ACK frame acknowledging receipt of configurations [Figure 1.3]. Now, Once the connection has been established, in case of no network activity, it automatically shuts-down after ICSL (Idle Connection State Lifetime) time that is decided during Client Hello. By-default its value is 30 sec. Since the server configuration has already been sent, an attempt at connection after passing of ICSL time does not require another handshake hence resulting in 0 RTT re-connection time [Figure1.5].
This can be seen from a captured packet flow as seen in Wireshark:

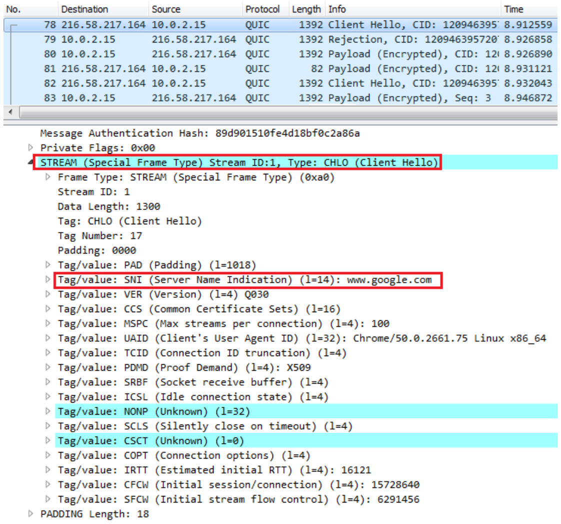
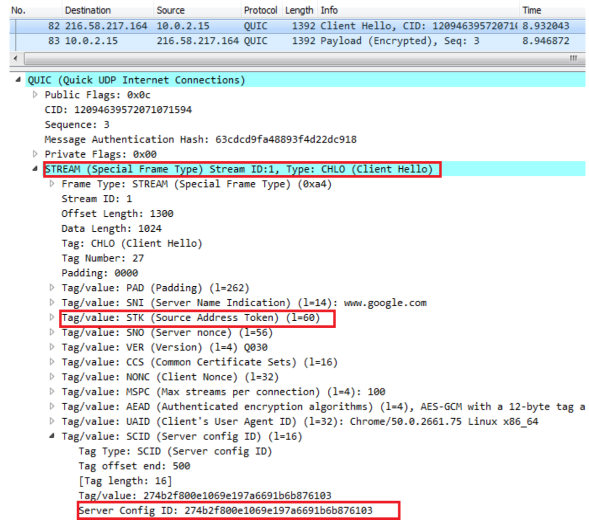


Figure 1.1 - Initial connection attempt by Client without any valid STK or SCID



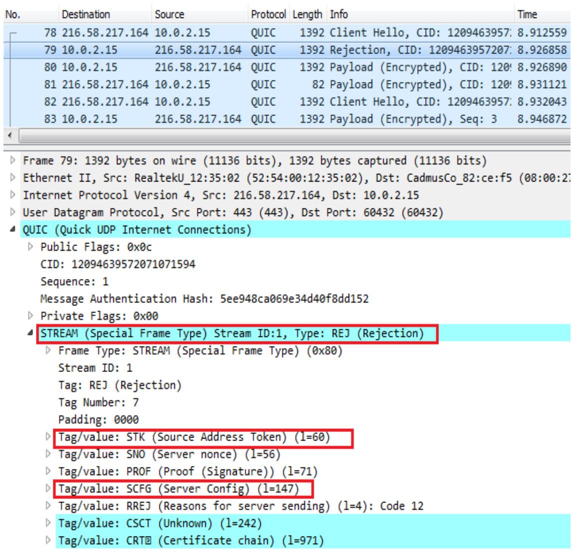


Figure 1.2 - Server Rejection Containing Valid STK and SCFG

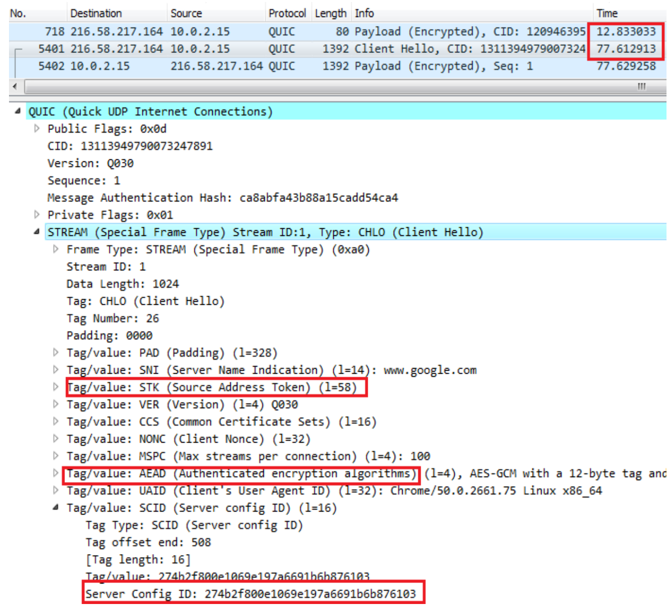


Figure 1.3 - New Client Hello with valid Server Config ID

Figure 1.4 New Connection Attempt after Expiration of ICSL

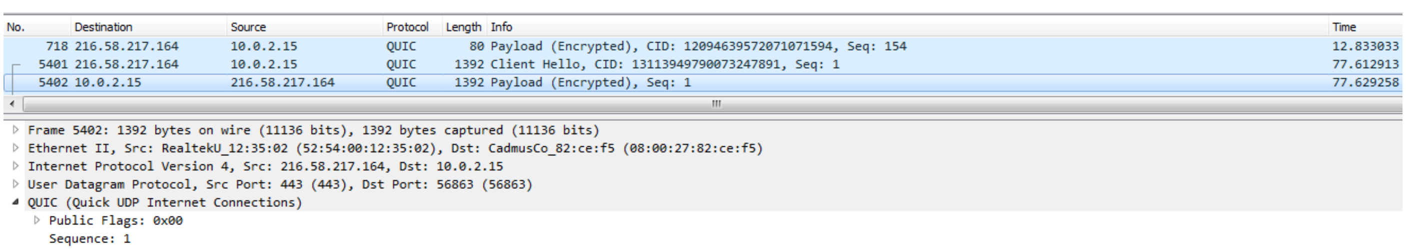


Figure 1.5 Server Responds without Rejection

Hence, 0 RTT Reconnection

Same observation has been displayed in our visualization tool as below:

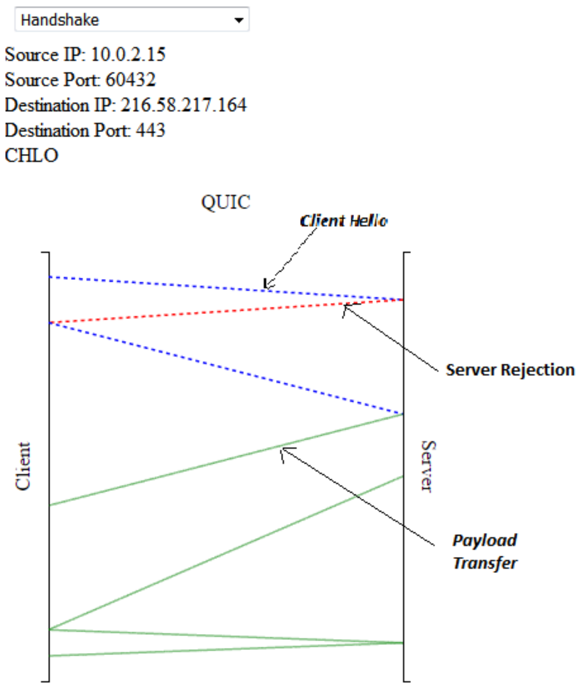


Figure 3.1: QUIC Handshake

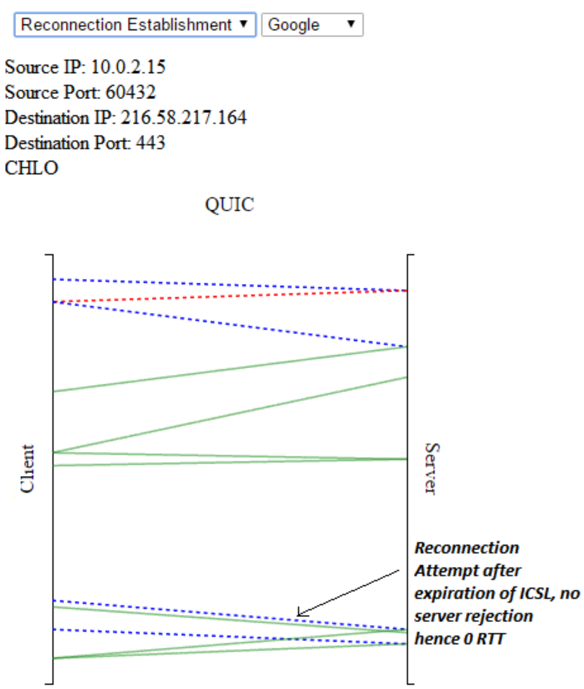


Figure 3.2: QUIC Reconnection

The blue dotted lines represent a connection attempt from client(CHLO) and red dotted line represents rejection by server (REJ). Green lines show payload transfer. On hovering over each line, the packet information such as source/destination IP, source/destination port number and whether it's a CHLO or server REJ is shown in the upper left corner of the display. Here, the first three flow lines show the 1 RTT initial handshake and during reconnection attempt there's no server rejection and hence 0 RTT.

5.2 Handshake and Reconnection in SPDY:

The packets captured were encrypted and marked as TLSv1.2 on Wireshark [Figure 2.1], but still the flow of packets could be observed manually. The initial handshake in SPDY requires 3 RTT. 1 RTT for initial TCP handshake and 2 more RTT for SSL handshakes. In case of reconnection, SSL handshake only needs to exchange keys between the client and the server and does not require server authentication thus reconnection establishment in SPDY requires 1 RTT for TCP handshake and 1 RTT for SSL handshake in needed. [Figure 2.2]

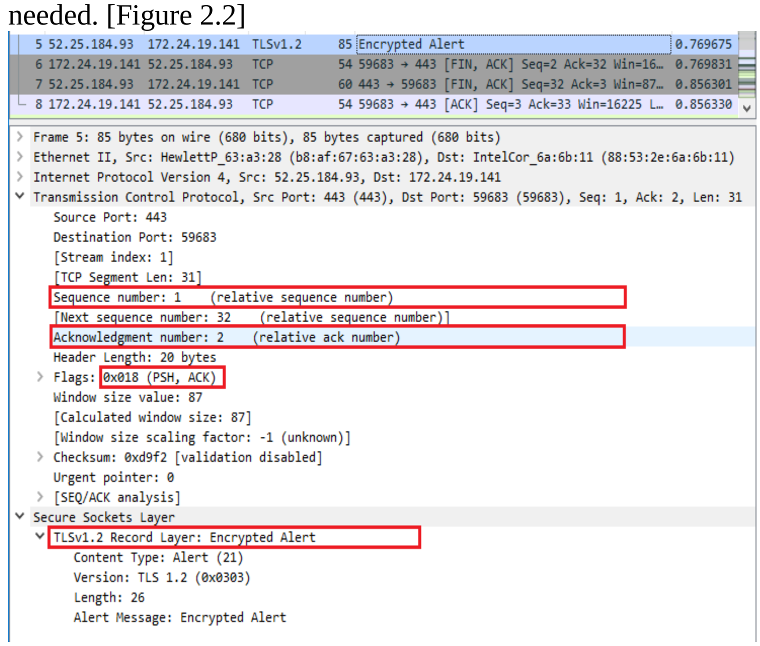


Figure 2.1 – Encrypted SPDY packet showing push acknowledgment

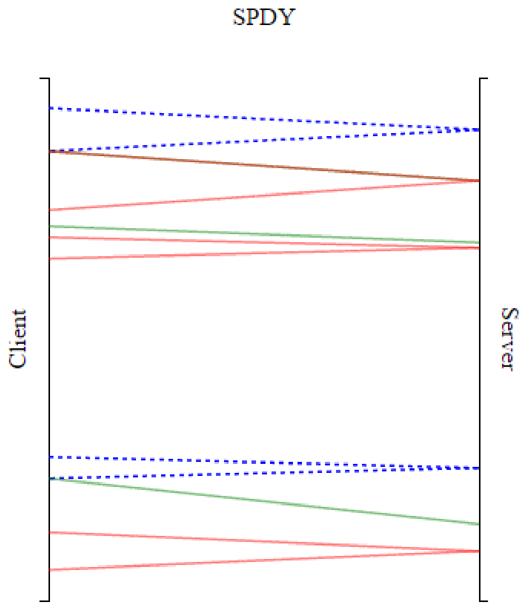
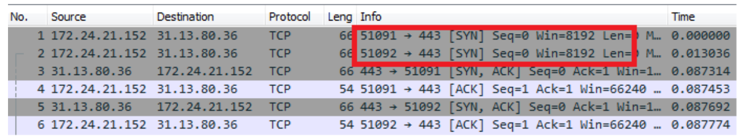


Figure 2.2 SPDY Handshake and Reconnection Flow as seen in Visualization Tool

In Figure 2.2, the blue dotted lines represent a TCP handshake. In the initial connection as well as the reconnection there is only one single TCP handshake. The handshake starts with the client sending a SYN and receiving a SYN ACK from the server. Following this a green line representing the ACK sent by the client in reply to the SYN ACK. This packet might be sent in parallel with the request for SSL server authentication represented by the brown line in the initial handshake and is followed by another handshake for SSL key exchange, represented by the red lines. As there is no need for server authentication in reconnection establishment the latter connection establishment only has one SSL handshake for key exchange.

5.3 Handshake and Reconnection in HTTP/1.1:

HTTP/1.1 uses three-way handshake (SYN-SYN/ACK- ACK) which requires 1 RTT. Along with being persistent it also uses parallel connections as can be seen in below wire-shark capture *[Figure 3.1]*. The reconnection attempt in case of HTTP/1.1 after a connection has been terminated requires the same three-way handshake process as the initial connection.



No.	Source	Destination	Protocol	Leng	Info	Time
1	172.24.21.152	31.13.80.36	TCP	66	51091 → 443 [SYN] Seq=0 Win=8192 Len=0 M…	0.000000
2	172.24.21.152	31.13.80.36	TCP	66	51092 → 443 [SYN] Seq=0 Win=8192 Len=0 M…	0.013036
3	31.13.80.36	172.24.21.152	TCP	66	443 → 51091 [SYN, ACK] Seq=0 Ack=1 Win=1…	0.087314
4	172.24.21.152	31.13.80.36	TCP	54	51091 → 443 [ACK] Seq=1 Ack=1 Win=66240 …	0.087453
5	31.13.80.36	172.24.21.152	TCP	66	443 → 51092 [SYN, ACK] Seq=0 Ack=1 Win=1…	0.087692
6	172.24.21.152	31.13.80.36	TCP	54	51092 → 443 [ACK] Seq=1 Ack=1 Win=66240 …	0.087774

Figure 5.1 Parallel Connections and Three-Way handshake in HTTP/1.1

This information can be seen in the visualization tool as below *[Figure 5.2 and Figure 5.3]*. The dotted blue lines represent SYN and corresponding SYN-ACK. Subsequent ACK and data flow can be seen by green lines. The upper left corner shows same information as for other protocols except for showing whether its SYN, SYN-ACK or ACK. It can be seen from the connected green lines that in HTTP/1.1 no new request is sent over a connection until it receives an acknowledgement for last one.

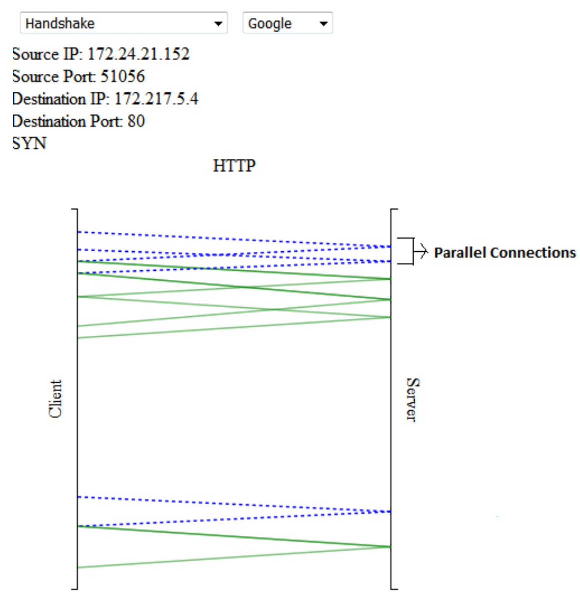


Figure 5.3 Handshake in HTTP/1.1

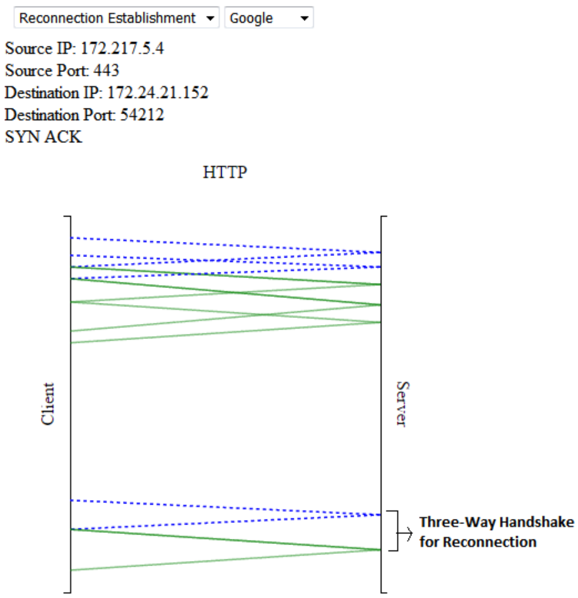


Figure 5.3 HTTP/1.1 Handshake and Parallel Connections

5.4 Page Load Time Comparison for Three Protocols:

Figure 6 shows the results as observed and displayed in the visualization tool:

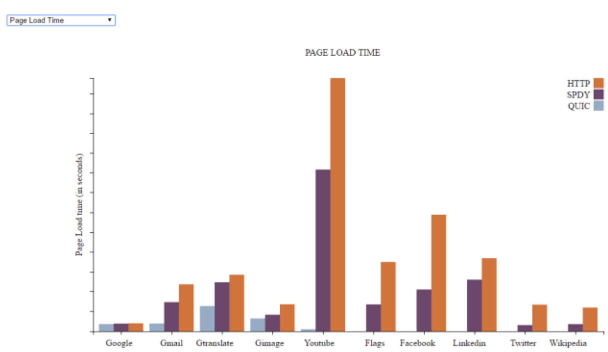


Figure 6 Page Load Time For Different Websites

It can be seen that QUIC performs better than SPDY and HTTP/1.1 wherever applicable especially in case of stream data (youTube.com). This can be because QUIC do not use packet sequence number while retransmitting it. This avoids retransmission timeouts by avoiding ambiguity about which packets have been received resulting in fewer re-buffers.

By looking at page load time for website *spdy.centminmod.com/flags.html* we can see that SPDY performs better than HTTP/1.1 when the page has large number of small objects as it combines multiple small http requests and response into one packet.

6. References

- QUIC Wire Layout Specifications - https://docs.google.com/document/d/1WJvyZflAO2pq77yOLbp9NsGjC1CHetAXV8I0fQe-B_U/edit#heading=h.z2ju224lr24y
- HTTP Over UDP: An Experimental investigation of QUIC - http://c3lab.poliba.it/images/3/3b/QUIC_SAC15.pdf
- Comparison of Web Transfer Protocols - http://proprogressio.hu/wp-content/uploads/2016/01/MolnarSandor_2015.pdf
- QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2 - https://tools.ietf.org/html/draft-tsvwg-quic-protocol-00