Does QUIC make the Web faster ?

Prasenjeet Biswal
Department of Computer Science
University of Houston

Omprakash Gnawali
Department of Computer Science
University of Houston

***Abstract*—Increase in size and complexity of web pages has challenged the efficiency of HTTP. Recent developments to speed up the web have resulted in two promising protocols, HTTP/2 (RFC 7540) at the application layer and QUIC (multiplexed stream transport over UDP). Google servers are using HTTP/2 and QUIC whereas other major sites like Facebook and Twitter have begun using HTTP/2. In this paper, we compare the performance of HTTP/2 vs QUIC+SPDY 3.1 by studying the Web page load times. In the first set of experiments, we serve synthetic pages (only static objects) over both protocols in emulated controlled network conditions and then extend it to real network, both wired and cellular (2G/3G in India and 3G/4GLTE in US). Further, we conduct experiments on a set of web pages on the most popular sites from the Internet (Alexa Rankings) in controlled conditions. We find QUIC to perform better overall under poor network conditions (low bandwidth, high latency and high loss), for e.g. more than 90% of synthetic pages loaded faster with QUIC in 2G compared to 60% in 4GLTE. This is due to the lower connection establishment latency and improved congestion control mechanism in QUIC. However, QUIC does not offer significant advantage when a webpage consists of many small-sized objects.**

1. INTRODUCTION

HTTP/1.1 is an application layer protocol, introduced in 1997 to transfer simple HTML pages over the Internet. The size and complexity of pages have increased multi-fold since then [1], [2]. While several modifications in protocols (like HTTP pipelining, TCP Fast Open), algorithms (data compression techniques), browser implementations and faster link speeds have helped to speed up the web, the base protocol has resisted efforts for a major upgrade until very recently.

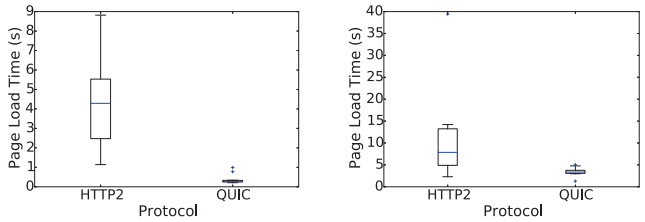
In 2012, Google proposed SPDY as an enhancement to HTTP/1.1 [3]. HTTP/2, which is largely based on SPDY, was standardized and published as RFC 7540 in May 2015. HTTP/2 provides several improvements over HTTP/1.1 like (1) multiplexed streams over a single TCP connection, (2) binary message framing for more efficient message processing, (3) header compression (HTTP/2 uses HPACK (RFC 7541) [4] compression), (4) server push thus eliminating the latency due to round trips due to client request, and (5) request prioritization.

HTTP/2 still has some performance limitations in the way it is used: head-of-line (HOL) blocking at the client due to TCP's in-order delivery and larger connection establishment time due to TCP's 3-RTT handshake (with TLS). Studies suggest that increasing the bandwidth past a point ($\sim$ 5 Mbps) does not reduce the page load time as the decrease in delay can [5]. QUIC (Quick UDP Internet Connections) was proposed in 2013 by Google to address these problems [6], [7]. QUIC's key features include (1) reduced connection establishment time (0-RTT) with security comparable to HTTPS, (2) improved congestion control - TCP Cubic + Packet Pacing with Forward Error Correction (FEC) packets (currently not enabled) to reduce retransmissions (but [8] indicates that FEC results in poor link utilization), (3) Connection Migration - QUIC uses connection identifier (CID) to uniquely identify a connection and eliminates reconnections in mobile clients during switch-over. QUIC promises improved multiplexing than HTTP/2 (which also solves head-of-line (HOL) blocking).

There has been interest in making QUIC part of the future standardization, thus studying its performance is important not only for understanding how protocol mechanisms perform in different circumstances but also to inform the Internet standardization bodies. This study informs the Internet community about the performance of QUIC so its design can evolve. Due to similarity in some mechanisms used by HTTP/2 and QUIC and the interest in borrowing mechanism [9], the insights from this study may also be applicable to HTTP/2. The study will also help web designers, network administrators, server and client engineers to design pages or systems or tweak network configurations to best utilize the features of QUIC and HTTP/2.

In this work, we compare the performance of QUIC and HTTP/2 and explain the reason for those differences. The measurement study is not trivial. The only QUIC server available is a sample server provided by Google, a basic in-memory server, and features like FEC are not implemented. Only a few browsers (Google Chrome, Chromium and Opera) support both HTTP/2 and QUIC. The page load time for a given webpage can be highly variable regardless of the server, especially under high loss (See Figure 1). The variations become more prominent when the page consists of scripts and stylesheets. Further, the measurements for

978-1-5090-1328-9/16/$31.00 ©2016 IEEE



(a) High Loss, Low Delay (b) High Loss, High Delay

Figure 1: High variance in Page Load Time under high loss, especially for HTTP/2.

complete web sites are specifically representative of network conditions at that time and the content of webpages. The results with these pages cannot be reproduced at different instances of time and for different users, because the number and size of embedded objects which are crucial in the performance study change rapidly and are personalized for sites like **cnn.com** or **amazon.com**. Therefore, we had to design web pages consisting of different combinations of object sizes and numbers for our experiments in controlled conditions.

We next review related work (§2) followed by extensive study of QUIC and HTTP/2's performance in different scenarios(§3). Finally we conclude and list the limitations of this work (§4). Throughout the paper, QUIC refers to QUIC as transport layer and SPDY as the application layer. Similarly, HTTP/2 means HTTP/2 as application layer over TCP as transport layer and SPDY means SPDY as application layer over TCP as transport layer.

2. RELATED WORK

Studies	HTTP	SPDY	HTTP/2	QUIC
[8], [10]	X	X		X
[11], [12]		X	X	
[13], [14]	X	X		
Our Study			X	X

TABLE 1: Previous studies comparing HTTP, SPDY, QUIC and HTTP/2.

QUIC studies: Although QUIC is relatively new, there have been few performance studies with QUIC. Carlucci et. al. [8] studied the impact of QUIC on goodput, link utilization and compared page load time for QUIC, SPDY and HTTP. The authors find that QUIC has better link utilization than HTTP (TCP), especially under high loss. Further, QUIC overall loads pages faster than HTTP and outperforms SPDY in high loss. However, FEC when enabled does not provide much benefit and reduces goodput (FEC packets use 33% of available bandwidth). In our work, we account for larger parameter space both for network conditions and composition of synthetic pages and use QUIC version 23 (compared to version 21) which increases the number of parallel streams from 6 to 110.

Somak Das studied page load times of Alexa's Top 500 sites [15] over QUIC, SPDY and HTTP/1.1 under different network configurations of bandwidth and delay [10]. The author reports that HTTP outperforms QUIC which outperforms SPDY on "very low bandwidth (0.2 Mbps)" link, but for low bandwidth links (0.3 - 1 Mbps) QUIC is better than HTTP/1.1. QUIC improves as RTT increases but is slower on high bandwidth and low-RTT scenarios. QUIC loads HTTPS sites much faster than HTTP/1.1. For mobile web, experiments are performed over cellular network traces and an enhanced congestion control protocol (Sprout-EWMA) is suggested. In our study, we additionally inject loss into the network to further study QUIC's performance in challenging environments. Also, we do an exhaustive study of performance on synthetic pages, which provides a better evaluation due to minimal dependency (among objects) and computation (stylesheets, scripts) compared to Alexa sites. We also perform comparisons across cellular networks spanning different technologies (2G, 3G, 4GLTE) and geographies (the US and India).

Unlike previous studies, we compare QUIC against HTTP/2, the new Internet Web standard (and likely to become a dominant protocol in the near future) and successor of SPDY (see the section on HTTP/2 vs SPDY below).

Google claims about 50% of the requests from Chrome to Google servers are handled over QUIC [16]. It reports QUIC works better than TCP on low-bandwidth connections, reducing the page load time for Google home page almost by a second on 1% of slowest connections. Over 75% connections have benefited from 0-RTT; even optimized websites have a 3% improvement in page load time. According to Google, benefits are more prominent in video services like YouTube (User feedbacks report 30% less buffering).

SPDY Studies: Wang et al. compare HTTP/1.1 and SPDY [14]. They use Apache's `mod_spdy` module for SPDY and regular Apache for HTTP/1.1. The initial experiments consist of serving synthetic pages with SPDY and HTTP under different network conditions. They found that SPDY is better for small objects and many large objects under low loss. They state that the main contributor to SPDY's better performance is due to its multiplexing over single TCP connection. This reduces total connection establishment time and results in fewer retransmissions (concurrent connections in HTTP/1.1 compete and induce loss), but the same feature prove detrimental under high loss due to aggressive congestion control mechanism (In HTTP/1.1, only 1 of 6 TCP connections is affected). Further they conduct experiments for real web pages (Alexa's [15] top 200 websites) using a self-designed module Epload to emulate page loads to get rid of browser dependencies and computations. They found that SPDY loads 70% of the websites faster over different network conditions.

A study has suggested that SPDY is not very beneficial over HTTPS and HTTP [13]. Their results suggest that SPDY is only 4.5% faster than HTTPS and 3.4% slower than HTTP connections. Also the median acceleration of SPDY over HTTPS was only 1.9%. They identify the main reason for low improvement of SPDY over HTTP(S) to be domain

Parameters	Range	High
bandwidth	1Mbps, 100Mbps	100Mbps
round trip time	10ms, 100ms, 200ms	≥100ms
packet loss (%)	0, 1, 2	≥1
object size	100B, 1K,10K, 100K	≥10K
#objects	2, 8, 16, 64, 128	≥64

TABLE 2: Factors for controlled experiments.

Type	Freq.	Core	Mem.	OS	Kernel
Client	1.2 Ghz	8	8 GB	14.04 (64-bit)	3.13.0-45
Server A (Controlled)	800 Mhz	2	4 GB	14.04 (64-bit)	3.13.0-44
Server B (Uncontrolled)	2.66 Ghz	4	8 GB	14.04 (64-bit)	3.13.0-44

TABLE 3: Server and Client Machine characteristics

sharding (SPDY works better for a single host website but most real-world websites download resources from different domains) and loading dependency of the web resources. Negative effects of domain sharding on SPDY performance is also discussed in [17].

Zaki et al. [18] compare SPDY with HTTP over 42 websites in Accra, Abu Dhabi, Bremen and New York. They report that SPDY performs better than HTTP in low-bandwidth networks (e.g. Accra).

HTTP/2 vs. SPDY: A comparison between raw HTTPS, HTTP/2 and SPDY/3.1 is discussed in [11]. They used HttpWatch [19] with Firefox to run simple page load tests on Google's UK homepage (https://www.google.co.uk/). They find that HTTP/2 packets have significantly small header size compared to SPDY because of HPACK header compression in HTTP/2. SPDY in turn has smaller header compared to HTTPS (HTTPS does not use header compression). The response body size of HTTPS is largest followed by HTTP/2 followed by SPDY. The larger response body of HTTP/2 compared to SPDY is a result of padding in data frames for greater security. The number of connections established are same for HTTP/2 and SPDY, in both cases fewer than with HTTPS. The page loads with HTTP/2 is faster than SPDY while HTTPS lags far behind due to larger connection establishment time. HTTP/2 outperforms SPDY due to smaller GET requests (HPACK compression) over asymmetric links. Other than the pointers discussed in [11], [12] states that HTTP/2 also benefits from multi-host multiplexing, improved prioritization and faster encrypted connections (ALPN extension instead of NPN).

To the best of our knowledge, this paper is the first work comparing QUIC with HTTP/2.

3. PERFORMANCE EVALUATION

3.1. Experimental Setup

We compare QUIC and HTTP/2 under controlled and uncontrolled environments. Table 2 provides the complete list of web page composition and network conditions for controlled experiments. To emulate different network

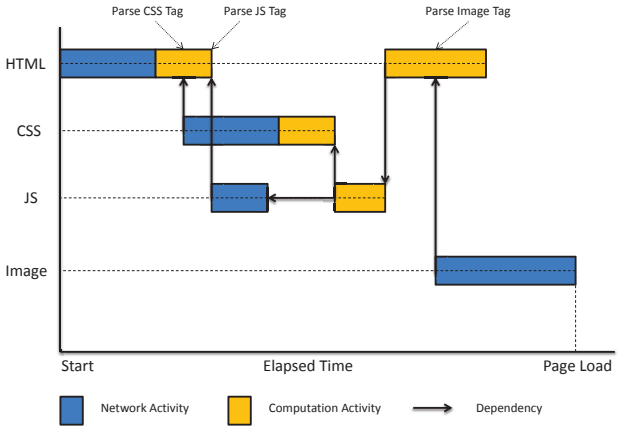
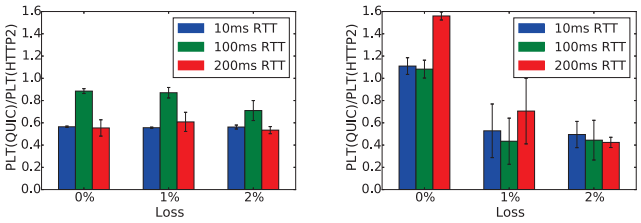


Figure 2: Different steps in a typical web page load. Reproduced partly from [14].



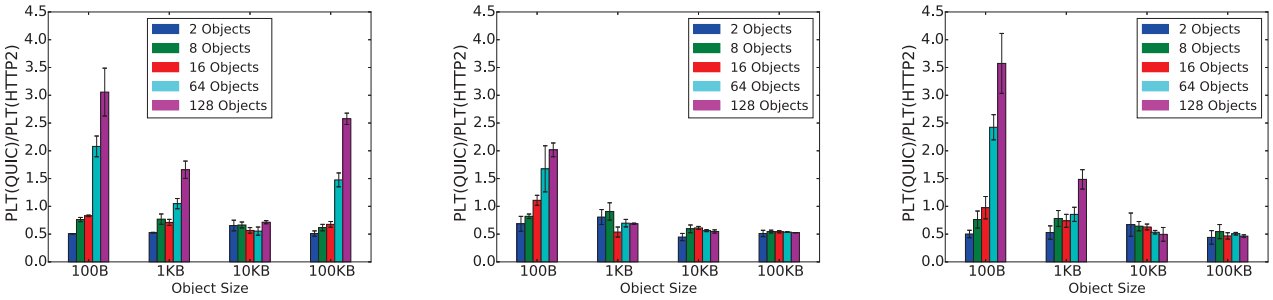
(a) 1 Mbps, 10KB, 64 Objects (b) 100Mbps, 10KB, 64 Objects

Figure 4: PLT(QUIC)/PLT(HTTP/2) for different combinations of loss and RTT.

conditions, we use Netem [20].

We use QUIC server (version 23) provided by Google. OpenLiteSpeed (version 1.3.9) [21] is used for handling HTTP/2 (draft-17) requests. We select OpenLiteSpeed because its HTTP/2 server is written in C++ (same as QUIC toy server) and supports the latest HTTP/2 draft-17. For each page load, we perform the experiment five times and take the median value to account for variance in measurements. Table 3 lists the client and server machine configurations. For controlled experiments, Server A and client machine were connected through a router. For uncontrolled experiments (both wired and mobile networks), a publicly accessible Server B was used. Chromium Browser v41.0.2272.76 is used as client for all experiments. Our goal is to compare the protocol performance within (not across) controlled and uncontrolled environment, so the difference in characteristics of the two servers does not impact the conclusions from this study.

To measure page load time, we developed an extension for Chromium client. The extension measures the total time spent solely in network activity by adding the time difference between the receipt of last byte and queuing time of all resources. We consider longest interval if loading time of



(a) Conf-3 (latency=200ms, loss=0%) (b) Conf-7 (latency=10ms, loss=2%) (c) Conf-9 (latency=200ms, loss=2%)

Figure 3: PLT(QUIC)/PLT(HTTP/2) for varying object sizes and number of objects

Configuration	1	2	3	4	5	6	7	8	9
Loss%	0	0	0	1	1	1	2	2	2
RTT (ms)	10	100	200	10	100	200	10	100	200

TABLE 4: The nine configurations used in the controlled environment experiments comparing the protocols.

multiple resources overlap. The extension only sums up the time taken in loading resources and discards the time taken for script execution. This helps to mitigate the variance in page load time to some extent. Fig. 2 depicts a typical download of a web page. The page load time (PLT) in this scenario is the time spent to download HTML, CSS and Image; JS download time does not contribute to the total page load time in this example. PLT also does not include the time taken to execute CSS, JS and some portion of HTML. The extension issues a xmlHTTPRequest to the Apache server running on same machine (client) which writes the measurements to a file. **Metric:**. We use the performance metric PLT(QUIC)/PLT(HTTP/2) to compare the protocols, where PLT stands for Page Load Time. If the page load is faster with QUIC than with HTTP/2, the ratio will be smaller than 1.0.

3.2. Synthetic Pages over Controlled Environment

We conduct experiments over the complete parameter space provided in Table 2 and the configurations listed in Table 4. We organize the main results from the performance comparison so we can understand the impact of these two factors on the page load times:

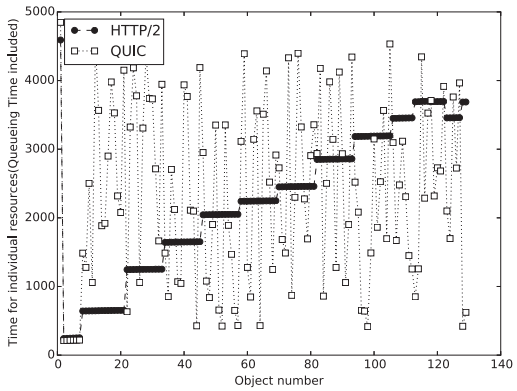
Impact of Page Content on QUIC's Performance: The number of objects and the size of objects affect the performance of protocols. Modern webpages have evolved to not only contain a range of object sizes but also a large range in terms of number of objects per page. Hence, we evaluate the protocols over a range of object sizes and the number of objects. Figs. 3(a), (b) and (c) show that as the number of objects increase (especially if the object size is small), HTTP/2 starts to outperform QUIC in terms of speed. However QUIC improves relatively over HTTP/2 as objects become larger and loss is introduced (see figs. 3 (b) and 3 (c)). The most suitable explanation for this condition is the head of line blocking experienced by HTTP/2 and efficient congestion control in QUIC.

Impact of Network Characteristics on QUIC's Performance: QUIC is designed for fast connection establishment and combating lossy networks. Consider the figure 4(b). The first group represents perfect conditions for HTTP/2 - high bandwidth, 0 loss. As expected HTTP/2 performs better than QUIC. But when loss is introduced, QUIC performs better than HTTP/2 as shown in the second and third group of bar plots. One question arises - Why does the performance of QUIC not improve when RTT is increased to 200ms (first group) even though QUIC has 0-RTT connection establishment? Here is a possible explanation - Though the 0-RTT connection establishment adds to QUIC's performance, it is prominent only on pages with a few small-sized objects (See graphs for 2, 8 and 16 objects in figs. 3 (a), 3 (b) and 3 (c)). On pages with large-sized objects, time spent in establishing connection is a tiny fraction of the time in loading the objects. QUIC performs better than HTTP/2 in lower bandwidth conditions. In contrast to figure 4(b), in figure 4(a), even the first bar group representing no network loss is less than 1.

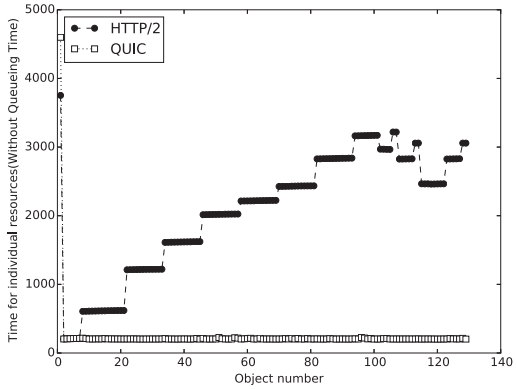
Summary of results over the complete parameter space: Through extensive experiments, we find that QUIC beats HTTP/2 under the following conditions:

1) the bandwidth decreases. QUIC is faster in 78% cases for 100Mbps link compared to 85% for 1Mbps. This is a marginal victory for QUIC as compared to other scenarios.
2) the size of the objects increase. For pages consisting of large objects (10KB and 100KB), QUIC performs better in 96% cases (95% for 100Mbps connection and 97.7% for 1Mbps connection.)
3) loss is injected into the network. Overall QUIC is faster in 82% of cases when loss is introduced (80% for 100 Mbps and 85% for 1 Mbps connection).

HTTP/2 loads a page faster when page consists of a large number of small-sized objects and there is minimal loss ($\sim$0%) in the network. Logically this condition should have been ideal for QUIC as this page composition can exploit



(a) Queuing Time included



(b) Queuing Time not included

Figure 5: Load time of resources. Configuration : 1Mbps, #Objects 128, Object Size 1KB, loss 0%, delay 200ms

the far more efficient multiplexing available in QUIC. To understand this contradictory behavior, we conducted experiments with web page consisting of 128 objects of one kilobyte under 1Mbps connection with a delay of 200ms and loss of 0%. As can be seen from figures 5 (a) and (b), for QUIC, the actual resource load time (time between request of resource to receipt of last byte) is very small compared to HTTP/2, whereas the total time to fetch each object once it is parsed in html is larger for many objects. One possible reason for this is that the QUIC toy server is not optimized to be used efficiently with Chromium browser.

To summarize, QUIC is faster (1.2-4.5X) when pages consist of few small-sized objects. QUIC also performs better when the page consists of large-sized objects. QUIC can achieve faster page loads in poor network conditions of lower bandwidth, high delay and lossy network. However, Page load with QUIC is slower (1.1-3.2X) when page consists of many small-sized objects.

3.3. Synthetic Pages over Wired Networks

We repeat the experiments over wired network on UH Campus LAN. HTTP/2 and QUIC servers run on publicly accessible machine. The difference from controlled environment is that bandwidth, loss and delay in this case are

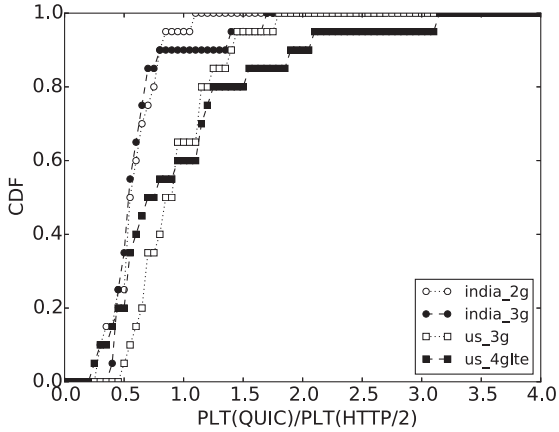


Figure 6: Performance of synthetic pages with QUIC and HTTP/2 under different cellular networks with clients in India and the US.

what is available on the live campus network. During the experiments, the network had a Median Bandwidth of 51 Mbps, Median Delay of 31 ms, and Median Loss of 0%. We find that the results from these experiments are similar to the results from controlled experiments under configurations 1 and 2 (0 loss and 10ms and 100ms latency).

3.4. Synthetic Pages over Cellular networks in the US and India

We run experiments with four different settings:

1) Client runs in India with Airtel 2G data enabled. We observed a network datarate of 170 Kbps and a delay of 334 ms.
2) Client runs in India with Airtel 3G data enabled. We observed a network datarate of 7.78 Mbps and a delay of 153 ms.
3) Client runs in the US with TMobile 3G data enabled. We observed a network datarate of 3.54 Mbps, loss of 0.048%, and a delay of 94ms.
4) Client runs in the US with TMobile 4GLTE data enabled. We observed a network datarate of 19.16 Mbps, loss of 0%, and a delay of 32ms

Figure 6 shows that more than 90% of pages have lower page load times with QUIC than with HTTP/2 when accessed from India. Compared to this only 60% of pages are faster over QUIC when client is in the US (low delay, high bandwidth, low loss). Also QUIC performs better in poor network conditions (2G in India and 3G in US) compared to their higher-speed counterparts (3G in India and 4GLTE in the US). This reaffirms, using real-world studies in two different countries, the earlier result of QUIC performing better in high RTT and lossy network.

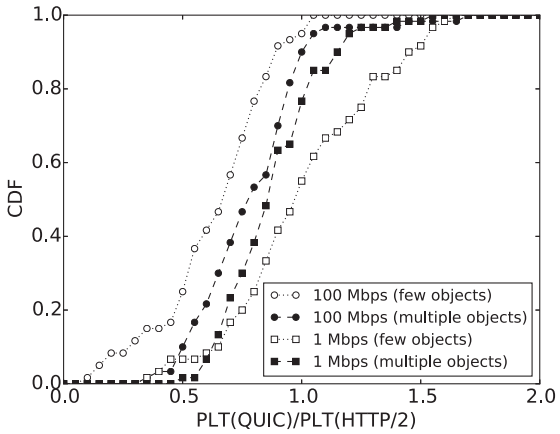


Figure 7: Performance of QUIC for complete web sites.

3.5. Complete Websites over Controlled Environment

Finally, we conduct our experiments on complex web pages consisting of images, stylesheets and scripts. We download top 200 sites from Alexa top sites [15]. From within this set, we selected 15 sites with small ($\leq$20) and other 15 with large (80-100) number of objects. Experiments are performed over four configurations (1, 3, 7 and 9 as shown in Table 4) with loss and delay varying from low to high.

Figure 7 shows that the maximum value of PLT(QUIC)/PLT(HTTP/2) drops to 1.6 from 3.2 (in synthetic pages). This is because the script execution time effectively serializes the loading of multiple objects on a page as the objects to be fetched are computed by the script. Also contrary to assumption, even dense pages (large number of objects) load faster over QUIC. The reason for this is that average page size is 2.5 MB and average object size is 25.39 KB (QUIC works better with large objects).

4. CONCLUSIONS

QUIC (stream multiplexed transport over UDP) was developed to speed up Web page loading, features 0-RTT connection, improved congestion control and multiplexing without head-of-line (HOL) blocking. Our experiments over controlled environment suggest that QUIC loads pages quicker than HTTP/2 under poor network conditions characterized by low-bandwidth, high delay and high loss. QUIC performance also improves with increase in object size. Increase in the number of embedded objects impacts QUIC's performance due to inefficient browser queuing. Our experiments over uncontrolled conditions (wired and cellular networks in the US and in India) confirm these observations. Experiments over complete web sites show performance gain with QUIC even for large number of embedded objects due to their large size.

Limitations: First, the only available QUIC server, which we used in this study, is not production-ready compared to OpenLiteSpeed's HTTP/2. Second, we do not quantitatively evaluate the effect of dependencies and computation on loading of embedded objects for complete web sites. Third, we did not conduct experiments on dynamic pages that are dependent on server processing.

References

[1] "Average Web Page breaks 1600k (jul. 18th 2014)," http://www.websiteoptimization.com/speed/tweak/average-web-page/.

[2] T. Everts, "Web Performance Today (Jun. 5th 2013)," http://www.webperformancetoday.com/2013/06/05/web-page-growth-2010-2013/.

[3] "SPDY Protocol Draft." https://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00.

[4] "HPACK: Header Compression for HTTP/2," https://tools.ietf.org/html/rfc7541.

[5] I. Grigorik, "Latency: The New Web Performance Bottleneck," https://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/.

[6] "QUIC: Design Document and Specification Rationale." https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/mobilebasic?pli=1.

[7] "Quic Wire Layout Specification." https://docs.google.com/document/d/1WJvyZflAO2pq77yOLbp9NsGjC1CHetAXV8I0fQe-B_U/edit?pref=2&pli=1.

[8] G. Carlucci, L. De Cicco, and S. Mascolo, "Http over udp: an experimental investigation of quic," 2015.

[9] "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2 draft-tsvwg-quic-protocol-02," https://tools.ietf.org/html/draft-tsvwg-quic-protocol-02.

[10] S. R. Das, "Evaluation of QUIC on Web Page Performance," Master's thesis, Massachusetts Institute of Technology, 2014.

[11] "A Simple Performance Comparison of HTTPS, SPDY and HTTP/2 (16th Jan. 2015)," https://blog.httpwatch.com/2015/01/16/a-simple-performance-comparison-of-https-spdy-and-http2/.

[12] J. Dorfman, "The Shift from SPDY to HTTP/2 (Mar. 2nd 2015)," https://www.maxcdn.com/blog/spdy-http2-shift/.

[13] G. Podjarny, "Not as SPDY as You Thought (Jun. 12th 2012)," http://www.guypo.com/not-as-spdy-as-you-thought/.

[14] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How Speedy is SPDY?" in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 387–399.

[15] "Alexa : The Web Information Company." http://www.alexa.com/topsites/countries/US., 2015.

[16] "A QUIC update on Google's experimental transport." http://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html.

[17] J. Graham, "Using CloudFlare to mix domain sharding and SPDY (26th Dec. 2013)," https://blog.cloudflare.com/using-cloudflare-to-mix-domain-sharding-and-spdy/.

[18] Y. Zaki, J. Chen, T. Pötsch, T. Ahmad, and L. Subramanian, "Dissecting Web Latency in Ghana," in *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 2014, pp. 241–248.

[19] "HTTPWatch," http://www.httpwatch.com/.

[20] "NeteM : Linux Network Emulator," http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

[21] "OpenLiteSpeed." http://open.litespeedtech.com/mediawiki/.