2018 IEEE Symposium on Computers and Communications (ISCC)

QUIC and TCP: A Performance Evaluation

Késsia Nepomuceno
Centro de Informática
Uni. Federal de Pernambuco
Recife, Brazil
ktcn@cin.ufpe.br

Igor Nogueira de Oliveira
Centro de Informática
Uni. Federal de Pernambuco
Recife, Brazil
ino@cin.ufpe.br

Rafael Roque Aschoff
Dept. de Ensino
Inst. Federal de Pernambuco
Palmares, Brazil
rafael.roque@palmares.ifpe.edu.br

Daniel Bezerra
Centro de Informática
Uni. Federal de Pernambuco
Recife, Brazil
db@cin.ufpe.br

Maria Silvia Ito
Centro de Informática
Uni. Federal de Pernambuco
Recife, Brazil
msi@cin.ufpe.br

Wesley Melo
Centro de Informática
Uni. Federal de Pernambuco
Recife, Brazil
wdbm@cin.ufpe.br

Djamel Sadok
Centro de Informática
Uni. Federal de Pernambuco
Recife, Brazil
jamel@gprt.ufpe.br

Géza Szabó
Traffic Laboratory
Ericsson Research
Budapest, Hungary
geza.szabo@ericsson.com

***Abstract*—Current Internet transport protocols are being revisited in an attempt to respond to traffic growth in size, diversity and the emergence of new applications. Designed to reduce Web TCP latency and connection establishment time, QUIC uses UDP and defines its own congestion control. To verify some of QUIC's performance claims, we carried out an extensive set of controlled experiments reflecting Internet traffic conditions by configuring various parameters, such as the round trip time (RTT), the packet loss ratio, web pages, and caching. Surprisingly, our experimental results show that QUIC presented worst page load time (PLT) results than TCP. We next present some of our interpretations of these results.**

***Index Terms*—QUIC, TCP, Page Load Time**

I. Introduction

The Internet has maintained a constant rise of its user base and an even more significant traffic growth since its creation. According to a release of the Cisco Visual Networking Index™ (VNI) Complete Forecast, such trend is not going to change soon. In the year 2020, the report foresees a volume of IP traffic almost three times the one observed in 2015. Additionally, more than one billion new Internet users are expected to join the global Internet community, growing from three billion in 2015 to 4.1 billion by 2020 [1].

The inadvertent Internet growth requires some changes to its infrastructure as well as protocols and applications. Web navigation is a critical affected activity, considering how web pages are increasingly becoming overloaded with information, such as dynamic objects or simply more data.

Web navigation is mostly controlled or supported by the TCP and HTTP protocols. However, HTTP was designed in the early 90's. Understandably, the design of such protocols did not predict the transport of today's evolving traffic. To overcome some of the problems, middleboxes in the form of cache or proxy servers were introduced to reduce web page load times. Additionally, the emergence of wireless devices brought the need for faster access times.

In the context of web navigation, the time required to load a page is one of the most important factors that determine end-user experience and satisfaction. The page load time metric (PLT) impacts on the way a user navigates and influences the time it spends at a site. Users who are frustrated by a slow-loading site, tend to visit the site once, leave to never return [2]. Moreover, PLT seems to have an impact on how well users recommend websites to others [3]. In fact, some search engines, such as Google, use the page load time in their algorithms as one of the factors to display their results [4], thus guiding users to sites with lesser PLT.

As a result, PLT is of great importance in web browsing. However, it tends to increase more and more due to the growth of the amount and variety of Internet data. The Internet community strongly fears that this evolution may lead to the congestion collapse in TCP traffic [5].

In order to develop an alternative that improves the loading of web pages and turn the Internet faster, especially for wireless users, researchers from both industry and academy put forward two new protocols, QUIC and SPDY. They were created to replace TCP and the HTTP respectively. Ultimately, the SPDY specifications were used as the basis for HTTP 2.0 protocol design.

A number of studies have already analyzed facets of these protocols, including security for QUIC [6], and also compared their performance [7] [8]. However, none of these studies answers the question of which protocol performs better on today's Internet.

Even though these protocols have been designed to reduce latency on the Internet, the claims have yet to be thoroughly verified.

In this work, we make a performance comparison between QUIC and TCP in order to establish which one is faster under network conditions that reflect the Internet nowadays. Our experimental approach consists in comparing the page load time from websites using both protocols to transfer the web pages content, since it is important to know if the new protocols actually work as designed and meet the current demands of the Internet.

This paper is organized as follows: In section II we give a theoretical background about the Internet evolution, the transport and application protocols. In Section III, we discuss some papers that investigated the QUIC protocol and highlight their contributions. Section IV presents our evaluation scenario, describing the tools used and how the experiments were carried out. We also present the factors and metrics used and their importance. Section V presents the results of our experiments, followed by a discussion of such results in Section VI. Section VII presents a summary of lessons learned and concludes this study. Lastly, in Section VIII we comment our next steps in the investigation of the QUIC protocol.

II. Background

In this section, we discuss the main Internet transport protocols presenting some design limitations and respective solutions.

A. TCP and HTTP

TCP and HTTP are the two most commonly used Internet protocols to transfer data and information, but the Internet traffic nature has changed. It has become increasingly voluminous, dynamic and complex to process [9].

As discussed in Section I, PLT directly determines the user experience. For instance, if a user needs to make a purchase but the web page does not load quickly, this user will be more likely to give up and perhaps select faster online stores instead. Research in [3] claims that a single second delay in page load time has been shown to cause a 7% loss in profits and 11% fewer page views. For an online store earning $50,000 a day, that one-second delay adds up to more than $1 million in lost sales each year.

Hence, the page load time is very important and the TCP and HTTP protocols have a direct influence on it. It is worth remembering that Internet transport protocols were not designed with latency in mind. Although they have evolved since the creation of the World Wide Web, they are currently representing the bottleneck of new Internet connectivity [10] especially when considering wireless users offloading their applications on Cloud services.

We can list the following protocol issues that, nowadays, represent a problem for the “new Internet”:

- Single connection per request, a TCP segment cannot load more than one HTTP request or response;
- Only clients can initiate requests, not servers;
- Uncompressed and redundant headers;
- Relatively long connection establishment times. Note that these can reach 3 Round Trip Times (RTTs) when combined with SSL authentication.

Among the bottlenecks of HTTP and TCP protocols is the fact that they rely on multiple connections for concurrency. This causes a series of problems, including additional packets following back and forth to configure the connection, slow-start delays, and client connection rationing [10]. Currently, most sites limit users to six TCP connections. Google and other Internet organizations proposed two new protocols aiming at eliminating or minimizing some of the issues mentioned previously. Such protocols are developed to operate at the application/session layer and transport layer, respectively, QUIC (Quick UDP Internet Connections) and SPDY.

B. QUIC and SPDY

QUIC (Quick UDP Internet Connection) is an experimental transport protocol created by Google and designed to provide security and reliability together with reduced connection latency (up to 0-RTT) [11]. The implementation of QUIC started in 2012 with the motivation to reduce the RTT to zero and help SPDY, a protocol also developed by Google to replace HTTP. QUIC inherited some SPDY features and solved some of its problems, like head-of-line blocking and the lengthy three-way handshake.

QUIC combines the best features of UDP and TCP with security tools whose main objective is to minimize Internet latency [11] [12].

QUIC also implements non-UDP features, such as congestion control and retransmission. It is not possible to build QUIC over TCP because TCP is already implemented in operational system kernels and middlebox firmware. Thus, because of the structure around TCP, complex modifications are unlikely to occur [13].

SPDY is the application protocol developed by Google in 2009 as an alternative to add efficiency to the HTTP protocol and decrease the latency on the Internet. It uses features such as data compression and push notifications, which provides resources from the server before the client requests them [14]. SPDY aims for faster navigation through the use of multiplexing, priority, compression and TLS (Transport Layer Security) [15].

From a didactic point of view, we can say that SPDY is divided into two layers: the Framing layer and the Application layer. The Framing layer multiplexes the data with a reliable Transport layer, i.e. the Framing layer uses a single channel to send the data through the transport protocol. The Application layer describes the mechanisms in which SPDY overrides HTTP requests and responses.

SPDY, together with QUIC, intend to make the Internet faster by reducing page load time and solving the TCP and HTTP problems mentioned above. They implement the following characteristics to achieve such goals:

- Multiplexing: QUIC multiplexes several QUIC streams over the same UDP connection. A client can send multiple HTTP requests and receive multiple responses over the same UDP socket [8].
- Request Prioritization: Some Web objects are more important than others and thus should be loaded earlier. The protocols allow a client to specify the preferred order in which resources should be transferred [16] [17]. Examples include JavaScript and CSS objects which process other objects within the page.
- Server Push/Hint: push allows a server to arbitrarily send resources without an explicit request from the client. Alternatively, SPDY can send hints advising clients to pre-fetch content [18].

- Header Compression: Using header compression, it is possible to considerably reduce the amount of redundant header information, each time a new page is requested [12].
- Universal encryption

III. Related Work

Due to statements about QUIC and TCP protocols, such as (1) QUIC reduces latency when compared to TCP [13] and; (2) QUIC performance is at least as good as the TCP performance [19], the scientific community sought to evaluate the performance of communication protocols in order to verify such statements.

A study comparing QUIC, HTTP and SPDY was developed by [7]. Their results claim that none of these protocols is clearly better than the others. The authors state that the actual network conditions determine which protocol performs the best. The authors in [20] present a rigorous comparison among QUIC versions and conducted experiments with QUIC and TCP. They investigated both protocols in a real Internet environment. While both works present interesting findings, we could raise some questions regarding the methodology of the experiments. For instance, since a real Internet link was used in [7] and [20], the results could be influenced by the dynamic nature of the connection, thus affecting the comparison. Moreover, the characteristics of the evaluated link do not necessarily represent average conditions of the Internet. Another point is the use of Google's servers in [7], which could benefit the QUIC protocol.

In [8], QUIC was also compared with HTTP and SPDY. The study reports that QUIC presented a reduction in the page load time when compared to the HTTP protocol in a scenario without packet loss. Moreover, QUIC presented greater page load time when compared to SPDY in a scenario with packet loss. Although the packet loss ratio is an important parameter, it cannot be considered a determinant parameter to performance evaluation of the protocols, since there are factors such as RTT, which affects the network performance and was not taken into consideration for the whole experiment.

The authors in [21] compare QUIC and TCP over HTTP 2.0. Their results show QUIC is faster than TCP when there are small-sized objects in the pages. Our work compares QUIC over HTTP 2.0 and TCP over HTTP 1.1 as further explained in Section IV-A. Moreover, our results don't analyze the size of the objects as detailed also in Section IV-A.

Considering all the above, we propose new experiments that evaluate the performance of QUIC in nowadays Internet. Through a controlled environment, we could simulate current states of the Internet with important network characteristics.

IV. Methodology

We have configured our environment in such a way that it matches real world's Internet as much as possible. Through state-of-the-art research we have realized and established various configurations that influence and represent such Internet, we can see these configurations in Table I. In the next subsections, we detail our experimental setup as well as the adopted metrics and factors.

TABLE I: Experiment factors

Factors	Levels
RTT (ms)	{20; 100; 200}
Packet Loss Ratio	{0%; 0.5%; 1%; 2%}
Protocol	QUIC; TCP
Cache	Enabled; Disabled
Web Page	{Page#1; Page#2; ...; Page#100}

A. Testbed and experiments

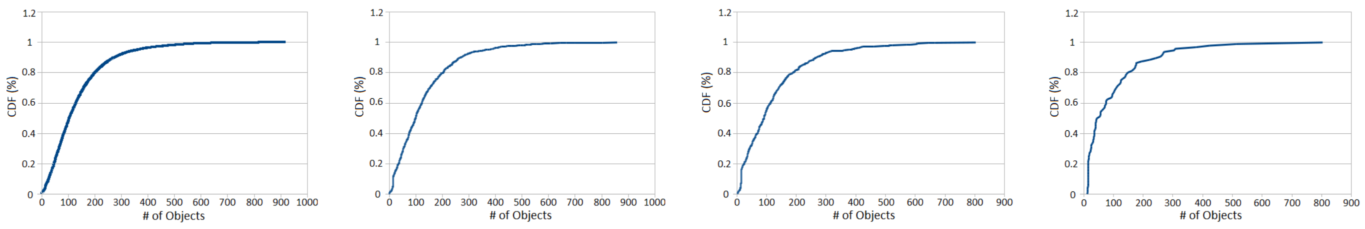
Our scenario was designed to reflect the real-world Internet traffic conditions as faithfully as possible. Therefore, we use the hundred most accessed pages in the world, according to the Alexa site ranking [22], to run our experiments. The top 100 Alexa sites present a better representation of the Internet as it displays the most accessed sites in the world and they are not restricted to a group. The web pages are from different servers, as page No. 1: Google.com; page No. 2: Facebook.com; page No. 3: Youtube.com; page No. 4: Baidu.com; page No. 5: Yahoo.com.

In order to decide on the top 100 pages, we picked the first ten thousand most accessed pages from Alexa and analyzed their number of objects distribution (Figure 1a). The idea of this initial step was to find a significant amount of web pages that would allow us to conduct our experiments within an adequate amount of time. Then, we kept lowering the number of pages to one thousand pages (Figure 1b), five hundred pages (Figure 1c) and one hundred pages (Figure 1d). Throughout our analysis, we noticed that the number of objects distribution kept the same behavior. As a result, we chose to run our experiments on the hundred most accessed web pages from Alexa.

To understand how their content is organized and to find the number of objects on Alexa's pages, we used the Google BigQuery tool. Google BigQuery contains a large database with web pages information. Thus, we uploaded Alexa's pages in Google BigQuery and, through a query, we extracted the number of objects from these pages, represented by the X-axis of the graphs from Figure 1a to Figure 1d.

With the 100 pages acquired, we created a script to record and replay the web pages using Mahimahi, which is a set of lightweight tools that record websites and replays them under emulated network conditions [23].

To reproduce the pages we configured the entire environment with virtual machines using the VMWare Player, an important tool to get a standard environment in all tests since it guarantees the same setup in the experiments. We use the Ubuntu 16.04 as the operational system. Pages were executed using Chrome version 54.0.28.40.100 as the client. Then, through Mahimahi, we run the pages with the values of packet loss ratio and RTT factors established; these configurations have four and three levels respectively, totaling 12 combinations of different configurations that were executed (see Table I). Each combination was performed thirty times



(a) 10,000 (b) 1,000 (c) 500 (d) 100

Fig. 1: Cumulative distribution of objects from Alexa site

for each of the hundred pages. We can see the combination of these configurations (C1-C12) in Table II.

TABLE II: Settings combinations between RTT and Packet Loss Ratio

Configuration	RTT	Packet Loss Ratio
Conf. 1	20 ms	0%
Conf. 2	20 ms	0.5%
Conf. 3	20 ms	1%
Conf. 4	20 ms	2%
Conf. 5	100 ms	0%
Conf. 6	100 ms	0.5%
Conf. 7	100 ms	1%
Conf. 8	100 ms	2%
Conf. 9	200 ms	0%
Conf. 10	200 ms	0.5%
Conf. 11	200 ms	1%
Conf. 12	200 ms	2%

We simulate the execution of these pages with the QUIC protocol enabled and disabled. Switching between those states, which is the main focus of our experiments, is possible through the web server Caddy, an open source web server written in Go [24]. At the time we performed the experiments, the FEC (forward error correction), the packet loss recovery technique, was disabled in Caddy because it was deactivated by QUIC [25].

As some of the recorded pages have resources that use HTTP and QUIC requires HTTPS, we used a proxy to redirect the traffic between the client and the server. To be fair and not influence the results, we force both QUIC and TCP experiments to connect via proxy.

Figure 2 summarizes the protocol configurations used during our experiments. At the Application layer, we used either HTTP 1.1 or HTTP 2.0. The former was combined with the TCP protocol, while the latter was used when analyzing the QUIC protocol. The protocols are combined like this considering that we are trying to compare the expected Internet structure (QUIC and HTTP 2.0) with the standard Internet (TCP and HTTP 1.1).

In order to be completely sure that the QUIC protocol was being used when configured during the experiments, we decided to inspect the data using tcpdump tool. This step is necessary because sometimes, while a request is made to use the QUIC protocol, it may not actually receive the go ahead and may instead use TCP.

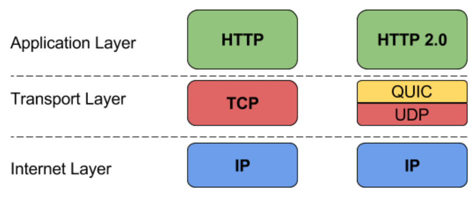


Fig. 2: Protocol scenarios

To acquire the page load time, we capture the HTTP archive record (HAR) for each of the hundred pages under all combinations of factors. A HAR format provides detailed records of the actions performed by the browser in loading the page [26]. To capture the HAR we used the Xdotool tool that was responsible for simulating keyboard and mouse movements and captures the file on the web page [27]. We collected two important events from the HAR file, the onContentLoad event, and the onLoad event, for each protocol we obtained a total of 72.000 results from both events.

For each event, we averaged the values in each different configuration and obtained the final onLoad and onContenLoad value for each page. Due to the fact that the QUIC protocol can only start its operations after the first connection, we configure the experiments with the disk cache as either enabled or disabled. We further discuss this topic in Section VI.

B. Factors and Metrics

For our experiments, when we replay the recorded pages, we consider the following factors that affect the loading performance of the web pages:

- Round trip time (RTT) with 20ms (intra-coast), 100ms (inter-coast) and 200ms (3G link or cross-continent). Such values are suggested by [16];
- Packet loss ratio with 0%, 0.5%, 1% and 2%, studies carried out claim that Google servers have a loss ratio between 1% and 2% [16];
- The recorded pages. Instead of using object size and number of objects as factors, we replace them with Alexa's top 100 as discussed before;
- Cache enabled and cache disabled. All experiments were performed with the cache disabled and then all experiments were repeated a second time with the cache enabled;

- And finally, we varied the transport layer protocol themselves, QUIC or TCP.

Although the bandwidth is an important factor in network evaluations, we chose not to employ it in our work because of the experiments nature. The protocol congestion control window does not rise to the level of using the entire bandwidth, i.e. the transmission ends before the connection uses it completely. Therefore, a higher or lower bandwidth does not influence the results.

As mentioned in Section IV-A, we used the following HAR events as evaluation metrics:

- onLoad measures the time taken to start rendering content [26];
- onContentLoad gives the time when pages rendering is completed [26].

Considering that we are collecting results from about 100 websites under different network and protocol configurations, it would be hard to show precise information regarding particular pages. However, that is precisely what we are not interested in showing. Our focus is the analysis of the two protocols corresponding to the current and new Internet configurations.

V. Results

In this section, we present the results obtained from the experiments performed following the directives described in Section IV. The goal is to compare the loading time of web pages using both protocols, QUIC and TCP.

Starting with Figure 3, we show a first view of how better the TCP performed against QUIC when considering the onLoad and onContentLoad events. Each of the four graphs included in Figure 3 shows 12 lines (four packet loss ratios $*$ three round-trip time values - see TableII) presenting the difference between the collected average page load time for QUIC and TCP. Equation 1 details how the plotted values were computed, where Avg_Q and Avg_T are the average values for either onContentLoad or onLoad events for the QUIC and TCP protocols respectively, and Q_{t_n} and T_{t_n} are collected values for QUIC and TCP for each of the 30 trials. In summary, the X-axis on Figure 3 represents pages from the Alexa basis while the Y-axis is the difference in page loading time in milliseconds (QUIC-TCP).

$$Diff_{(quic-tcp)} = Avg_Q - Avg_T = \frac{\sum_{n=0}^{30}(Q_{t_n} - T_{t_n})}{30} \quad (1)$$

In Figure 3, by presenting the result as the difference between QUIC and TCP PLTs, we can focus on observing if the plotted lines are bellow or above 0 in order to verify which protocol performed better. By showing all twelve lines at once, we provide a better overall view of the performance difference between the protocols, on the other hand, we lose some fine-grained detail. Moreover, we decided to order the collected results from the lowest value (where QUIC performed better) up to the greatest value (i.e. when TCP performed better). While with this presentation we lose information regarding the specific page that produced the result, it is easier to see the overall picture. Besides, as we already mentioned, we are not interested in the performance of any particular page, but that of the protocols themselves.

Figures 3a and 3c present the results for the event onLoad with and without cache, respectively. Figures 3b and 3d show the results for the event onContentLoad with and without cache respectively. As we can easily see from the four graphs of Figure 3, regardless of the configurations, TCP performed better on at least 60% of the assessed pages of any configuration or metric.

A more detailed information including the percentage of pages from the basis where the page load time was better for QUIC than TCP is shown in Table III. As we can see, the number of pages where QUIC outperformed TCP ranged from 5.37% on the 9th. configuration with cache up to 40.62% without cache on both configurations 4 and 8. Considering the three parameters shown in the table (configuration, cache, and event) we are able to see high variations in the collected results. We can conclude that both the choice of caching, as well as the scenario (configuration) have a great impact on the collected results. For instance, by fixing onLoad, Cache Enable, and varying the configurations (second column of Table III), the results differ in more than 16%. Similarly, by fixing onContentLoad, Configuration 01, and varying cache, we can see that the QUIC protocol went from less than 10% of pages (cache enabled) which loaded faster using it to more than 30% (cache disabled).

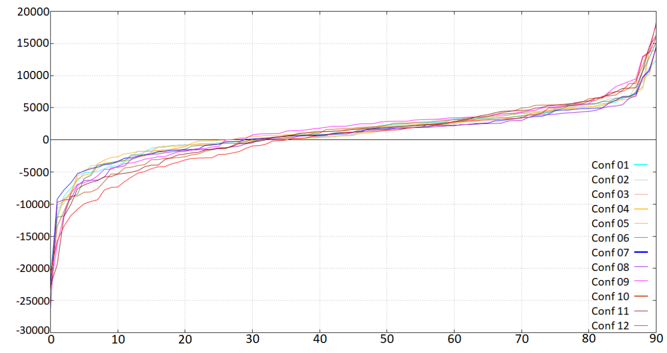
TABLE III: Percentages of pages for which QUIC performed better than TCP under setting cache enabled (CE) and cache disabled (CD)

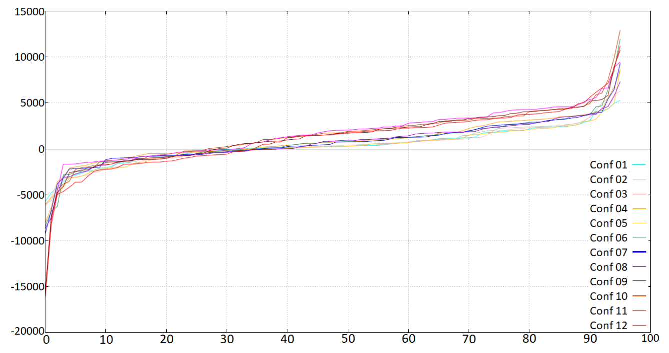
	onLoad		**onContentLoad**	
Conf.	CE	CD	CE	CD
01	17.24%	35.16%	9.67%	34.37%
02	22.98%	34.06%	12.90%	37.50%
03	24.13%	34.06%	8.60%	39.58%
04	16.09%	35.16%	4.30%	40.62%
05	25.28%	29.67%	11.82%	36.45%
06	21.83%	35.16%	7.52%	36.45%
07	27.58%	31.86%	10.75%	37.50%
08	26.43%	34.06%	10.75%	40.62%
09	20.68%	28.57%	5.37%	33.33%
10	24.13%	32.96%	8.60%	28.12%
11	25.28%	35.16%	9.67%	30.20%
12	32.18%	39.56%	9.67%	36.45%

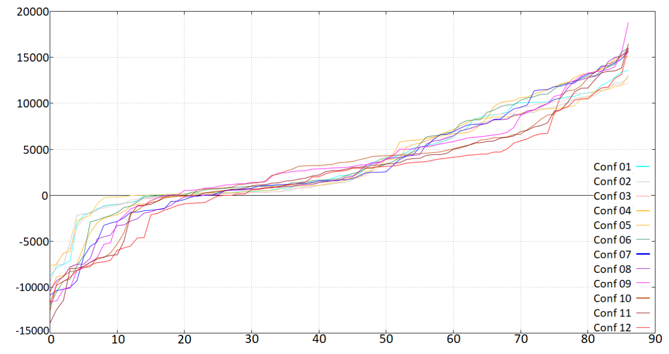
Finally, in order to have an idea of the dispersion of our data, we decided to include a boxplot of a slice of our experiments on Figure 4. More precisely, Figure 4 shows a boxplot for our results when considering the metric onLoad without cache. The y-axis represents the page load time metric of both protocols and the x-axis reflects the different configurations in which we run the experiments.

VI. Discussion

In this section, we provide additional discussion regarding our experiments as well as insights of what we believe could explain both the expected as well as the unexpected results.


(a) onLoad without cache


(b) onContentLoad without cache


(c) onLoad with cache

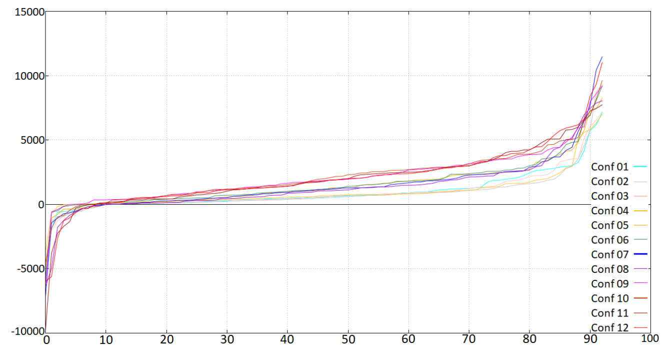

(d) onContentLoad with cache

Fig. 3: Page Load Time (QUIC - TCP) for onLoad and onContentLoad event with and without cache.

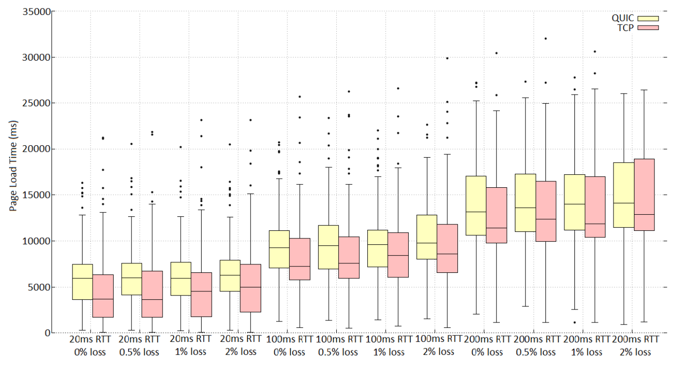


Fig. 4: PLT from QUIC and TCP for onLoad event in experiments with cache disabled

First of all, for both group of experiments running with or without cache, we observed that the RTT variation has a significant impact on the PLT metric. As depicted in Figure 4, by fixing the packet loss ratio and varying the RTT in 20 ms, 100 ms, and 200 ms, we can easily see the increase in the page load time. When we fix the RTT and vary the packet loss to 0%, 0.5%, 1%, and 2%, however, this hardly influences the results. These findings are in line with [28], which also shows RTT as having a big impact on the performance of QUIC when comparing with TCP while the loss of packets is not that meaningful.

Our first batch of experiments were actually the ones with cache disabled, and if we were to consider only this group of experiments, it is not hard to see that TCP performed better in most of the cases. As we can see in Table III, regardless of the configuration, the results show QUIC providing a better page load time in less than 40% of the web pages considered only.

While this is unexpected, a conceivable reason could be that QUIC is not the Internet mainstream protocol, hence, for a client to use QUIC in the first connection, it initially needs to run a discovery mechanism. For the first connection, the client uses TCP to send requests to a server. When a server supports QUIC, it informs the client and the remaining connection is maintained with QUIC.

In order to verify such assumption, we decided to run a set of experiments while enabling the cache system. More specifically, by using cache, the browser would beforehand know if the page can be served by QUIC and hence avoid the initial discovery mechanism. We can see the results in both Figure 3 and Table III.

To our surprise, we observed that enabling cache while using TCP produces a much higher performance improvement than when using QUIC. This is reflected by the number of pages that were loaded faster using QUIC to be even lower. We could mention possible reasons why the results were perhaps counter-intuitive for QUIC when enabling the caching system. For instance, the engine may not be as efficient to handle the cache with the QUIC protocol as it is with the mainstream protocol, the TCP.

Since we did not manage to identify any particular pattern that could directly categorize the pages that benefited more or less with QUIC or TCP, we sought to search for any patterns in the PLT phases (Wait, DNS, Send, Connect, Blocked, Receive), which are obtained from the HAR file [9]. We found out that the Connect Phase prevailed as the most time-consuming phase in the pages loaded with the TCP protocol, meaning that TCP loses time when creating connections to the server. Thus, as detailed in Section II-B, the multiplexing from QUIC turns to be a good solution to avoid this issue. However, in pages using QUIC, we noticed that the Receive Phase was the most time-consuming on most of the pages we could test, i.e. QUIC loses time when reading the server (or cache) answer.

Finally, one of the reasons that could also explain why our experiments show overall better results for TCP is the fact that our list of pages was not selected from those structured in a way that would theoretically perform better with QUIC. We instead chose what we consider to be a good representation of Internet pages, by considering the most accessed pages in the world, which are not necessarily optimized for QUIC yet.

VII. Conclusion

There are some community discussions on which protocol has the smallest page load time. The loading time of a web page directly affects the user's experience and will reflect on future actions of the Internet users.

Aiming at these discussions, this study compared the performance of QUIC with HTTP 2 and TCP with HTTP 1.1. We have included in this work the most accessed Internet pages in different network conditions to evaluate the performance of the protocols.

First, by analyzing the list of the most accessed pages in the world, we found out that the number of objects distribution remains similar in the different subsets of the list. Also, by measuring the loading time of the pages with the onLoad and onContenLoad events, in the described scenarios, we noticed that QUIC presents a worst performance than TCP under the respective application protocols.

Inspecting our results, we could observe an interesting behavior regarding our factors, also noticed in other studies [28]. As discussed in the previous section, the influence of the RTT in our experiments was noticeable while the packet loss ratio influence was inexpressive.

Although QUIC is already being used in part of the Internet, there are still some obstacles and improvement points until it becomes the mainstream Internet protocol. Once the gain in the performance is not so clear. As the Internet structure, e.g. servers distribution and web pages formats, is optimized for the current protocols and QUIC is still adapting, its performance could be inferior than the expected. Therefore, the full deployment of QUIC must still be discussed before the protocol adoption.

VIII. Future Works

As future works, we intend to continue the performance evaluation of the QUIC protocol. For more case studies, we have planned to set different levels of the factors studied here. We also planned to add new features of networks that would influence the metric of page load time.

References

[1] Cisco System Inc. Cisco visual networking index predicts near-tripling of ip traffic by 2020, 2016.

[2] Big Commerce. Big commerce. https://www.bigcommerce.com/blog/4-tips-improve-ecommerce-bounce-rate-right-now-sellmore-video/.

[3] PR Newswire. 'customers are won or lost in one second,' finds new aberdeen report, 2008.

[4] Google Inc. Using site speed in web search ranking, 2010.

[5] J. Widmer, R. Denda, and M. Mauve. A survey on tcp-friendly congestion control. *IEEE Network*, 2001.

[6] Robert Lychev, Samuel Jero, Alexandra Boldyreva, and Cristina Nita-Rotaru. How secure and quick is quic? provable security and performance analyses. In *Security and Privacy (SP), 2015 IEEE Symposium on*, 2015.

[7] Péter Megyesi, Zsolt Krämer, and Sándor Molnár. How quick is quic? In *Communications (ICC), 2016 IEEE International Conference on*, 2016.

[8] Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. Http over udp: an experimental investigation of quic. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015.

[9] Jan Odvarko, Arvind Jain, and Andy Davies. Http archive (har) format. *W3C draft*, 2012.

[10] Mike Belshe and Roberto Peon. Spdy protocol. *URL http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3-1*, 2012.

[11] Janardhan Iyengar and Martin Thomson. Quic: A udp-based multiplexed and secure transport. *draft-ietf-quic-transport-01*, 2017.

[12] Jeffrey Erman, Vijay Gopalakrishnan, Rittwik Jana, and Kadangode K Ramakrishnan. Towards a spdy'ier mobile web? *IEEE/ACM Transactions on Networking*, 2015.

[13] Google Inc. Chromium. https://www.chromium.org/quic.

[14] Nivedita P Regundwar, Dhanashri A Shukla, and Payal Lokhande. A studypaper on spdy protocol: Lets make the web faster [j]. *International Journal of Scientific and Engineering Research*, 2013.

[15] Jim Roskind. Quic, quick udp internet connections, 2013.

[16] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. How speedy is spdy? In *NSDI*, 2014.

[17] Bob Briscoe, Anna Brunstrom, Andreas Petlund, David Hayes, David Ros, Jyh Tsang, Stein Gjessing, Gorry Fairhurst, Carsten Griwodz, and Michael Welzl. Reducing internet latency: A survey of techniques and their merits. *IEEE Communications Surveys & Tutorials*, 2014.

[18] Yehia Elkhatib, Gareth Tyson, and Michael Welzl. Can spdy really make the web faster? In *Networking Conference, 2014 IFIP*, 2014.

[19] Florian Gratzer. Quic-quick udp internet connections. *Future Internet and Innovative Internet Technologies and Mobile Communications*, 2016.

[20] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. Taking a long look at quic: An approach for rigorous evaluation of rapidly evolving transport protocols. IMC '17. ACM, 2017.

[21] P. Biswal and O. Gnawali. Does quic make the web faster? In *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec 2016.

[22] Alexa Internet, Inc. Alexa top sites. http://www.alexa.com/topsites.

[23] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for http. In *USENIX Annual Technical Conference*, 2015.

[24] Light Code Labs. Caddy server. https://caddyserver.com/.

[25] Github. Pointers for implementing the client. https://github.com/lucas-clemente/quic-go/issues/310#issuecomment-253555550, August 2016.

[26] Michael Butkiewicz, Harsha V Madhyastha, and Vyas Sekar. Understanding website complexity: measurements, metrics, and implications. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, 2011.

[27] J Sissel. xdotool–fake keyboard/mouse input, window management, and more, 2013.

[28] S. Cook, B. Mathieu, P. Truong, and I. Hamchaoui. Quic: Better for what and for whom? In *2017 IEEE International Conference on Communications (ICC)*, 2017.