QUIC: Better For What And For Whom?

Sarah Cook, Bertrand Mathieu, Patrick Truong and Isabelle Hamchaoui
Orange Labs
Lannion, France
Email: {sarah.cook,bertrand2.mathieu,patrick.truong,isabelle.hamchaoui}@orange.com

***Abstract*—Many applications nowadays use HTTP. HTTP/2, standardised in February 2015, is an improvment of HTTP/1.1. However it is still running on top of TCP/TLS and can thus suffer from performance issues, such as the number of RTTs for the handshake phase and the Head of Line blocking. Google proposed the QUIC (Quick UDP Internet Connection) protocol, an user level protocol, running on top of UDP, to solve those issues. Google argues that the response time (Page Load Time) is shorter and thus the end-user experience better. First papers evaluated the intrinsic performances of QUIC, but none compared QUIC with the network, the website structure and the involved actors in mind. In this paper, we present the results of our evaluation, performed on a local testbed as well as on Internet, and our analysis to identify in which conditions QUIC is of interest, which actors can benefit from having QUIC deployed in the network and what impacts QUIC can lead to.**

I. Introduction

HTTP represents a huge portion of the Internet traffic. HTTP/1.1 is still used by many applications, but HTTP/2 standardised in February 2015 at the IETF [1] [2], is increasingly implemented by web servers and all current browsers now support it. Compared to HTTP/1.1, HTTP/2 has many advantages. One of them is the multiplexing of streams which optimises the delivery. However, still relying on TCP like HTTP/1.1, HTTP/2 suffers from the problem of Head-of-Line (HoL) blocking, which can affect the end-users experience in case of packet losses. Furthermore, TCP encrypts the data via the TLS protocol, which needs a handshake phase of 2 Round Trip Time (RTT) duration, in addition to the necessary 1-RTT TCP handshake. Setting up a first connection from a client to the server thus requires 3 RTT messages.

QUIC (Quick UDP Internet Connections) is a new network transport protocol by Google [3], which runs on top of UDP, instead of TCP, thus removing the need for the initial TCP handshake mechanism. It implements its own encryption system, comparable to TLS, which combines connection establishment and key agreement into only 1 RTT. However, most of the time, QUIC can start a connection in 0 RTT, immediately sending encrypted application data to the server, when it already has in its cache the server certificate (from a previous connection). Furthermore, running on top of UDP, QUIC avoids the HoL issue we can have with TCP in case of packet loss.

The performance evaluation of QUIC has been studied by a few research papers in the literature, mainly focusing on the use case of QUIC as a transport for HTTP. In [4], the authors use the QUIC toy server proposed by Google in the Chromium codebase to evaluate the CUBIC congestion control algorithm implemented in QUIC with the one implemented in TCP [5], and they also analyze the load times of a web page when using QUIC, HTTP/1.1, SPDY (a now-deprecated protocol by Google, which has served as a basis for HTTP/2 specifications). Paper [6] also proposes a similar evaluation of QUIC compared to HTTP/1.1 and SPDY, but they use Google servers to serve web pages over QUIC. Paper [7] presents a security evaluation of QUIC and analyzes how specific replay attacks could disrupt the cryptography handshake in QUIC, falling back to the default TCP transport as if the client does not support QUIC.

However, these evaluations are specific and limited in their scope and do not take into consideration the ecosystem. QUIC is currently only deployed on Google servers through its services (Search, Gmail, YouTube, etc.) and only Chrome and Chromium browsers natively support the QUIC protocol. But since Google is an important player, it represents a non negligible portion of the network traffic. For instance, for the past months, QUIC traffic has exceeded 5% of the total traffic in the Orange French backbone. It is then important to analyse the real behaviour of QUIC, compare it to other solutions, but also to identify which actors can be most impacted or who can benefit from a wide deployment of QUIC.

This paper presents our analysis of QUIC potentiality, mainly with regards to end-users, network and service providers based on our evaluation of the QUIC protocol. We compare QUIC (or more exactly HTTP/2 over QUIC [8]) to HTTP/1.1 and HTTP/2 (over TCP+TLS) on a local testbed, where we can easily change the network conditions (packet loss, delay) and website structure, and on the real Internet, where our client accesses the Youtube web servers. We first present in Section II an overview of the QUIC protocol and its main features. Then, we present in Section III the testbed we set up as well as the tools we use. In Section IV, we detail the main results of our performance evaluation of the protocols and their impacts on the actors involved in the delivery chain. We finally conclude this paper in section V.

II. Quick Overview of the QUIC protocol

Starting from 2012, Google has actively developed its experimental QUIC protocol, and a Working Group has recently been created at IETF [9] to push it as a standard for web content delivery. QUIC is a general-purpose transport protocol run over UDP, but built into the user space (Fig. 1), limiting the protocol deployment to endpoints only, without changing

978-1-4673-8999-0/17/$31.00 ©2017 IEEE

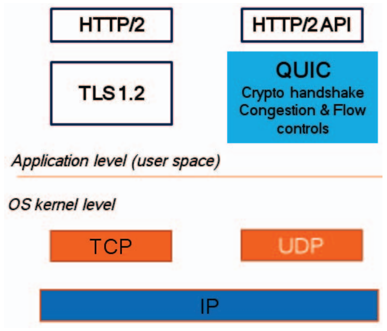

Fig. 1. QUIC Overview

network middleboxes. For the moment, Google mainly highlights the use case of QUIC as a transport for HTTP/2.

Relying on UDP, QUIC multiplexes streams without the head of line blocking downside inherent to TCP. Requests from a same client are processed in QUIC as streams that are multiplexed over a single UDP connection (identified by a unique 64-bit ID) between the client and the server, and any lost packet in a stream will not prevent the client from processing packets from other streams.

QUIC natively (and not optionally) enforces security through authenticated and encrypted packet header and payload. While providing all security guarantees similar to TLS, QUIC can start a connection and negotiate all the certificate logic in at most one network round-trip time (RTT), depending if it is your first connection to a new server or a repeated connection to an already visited server.

Low-latency connectivity of QUIC is mainly due to the use of UDP, and the price to pay is that QUIC has to build new mechanisms on top of UDP to make connections at least as reliable as in TCP. To this end, Google introduces mechanisms to make QUIC more resilient to congestion, packet reordering or loss than pure UDP. For example, in addition to using packet pacing to reduce packet loss and a Forward Error Correction (FEC) to reduce retransmission latency, TCP Cubic [5] has been reimplemented as the default congestion control in QUIC, but Google also gives users the freedom to plug their own congestion control algorithms. Regarding the flow control, QUIC provides a stream and connection-level mechanism equivalent to the HTTP/2 credit-based flow control. This means a QUIC receiver also uses $WINDOW\_UPDATE$ frames, first for stream-level flow control to advertise how many bytes it is willing to receive on a stream, and secondly, for connection-level flow control to limit the buffer used by the receiver to aggregate all streams on a same connection.

III. Testbed and tools

To evaluate the different transport protocols, we set up a testbed allowing us to make reproducible tests in a controlled environment (local testbed) and realistic tests, as any end-user at home, with a homebox and ADSL connectivity (remote testbed) (Fig. 2). We also developed a specific configurable and scriptable tool (perfy) to send HTTP requests to retrieve any web page using the desired protocol (HTTP/1.1, HTTP/2 or HTTP/2 over QUIC).

A. Testbed

Our main goal is to evaluate the performance, especially the Page Load Time (PLT) [10], of websites using the three aforementioned protocols (HTTP/1.1, HTTP/2, HTTP/2 over QUIC) under various network conditions. To this effect we set up a two-sided testbed offering both a local and a real world Internet connectivity.

1) Local Testbed: In the local testbed, we installed a machine, with Docker as a virtualisation system. This enables us to deploy several web servers on the same host, each one being isolated and running independently in its container. The QUIC web servers are based on the Go-Quic implementation [11], and the HTTP servers use the HTTP Golang libraries. For the contents, the web servers we used are replicas of public websites (Youtube, Orange, Doctor ANR project). For each of them, their contents can be hosted on a single web server (one Docker container) or distributed amongst several web servers (several Docker containers). This allows us to evaluate if the protocol is better when contents are co-localised or distributed on several servers.

To evaluate the behaviour of the protocols when network conditions change, a network emulator is set up by using the netem functionality of the traffic control (tc) Linux command, to introduce delay and packet losses on the link between the client and the servers. This network emulator is fully configurable (delay, percentage of packet loss, and the related IP sessions if desired). By setting different values of loss and/or delay on the link to the separate docker containers, each with its own IP address, it is possible to simulate web servers in different geographical locations.

Finally, the client is a PC where the *Perfy* tool we developed is installed (see section III-B).

2) Remote Testbed: For the remote testbed, we used the same client PC and tool, but connected directly to our homebox (Orange Livebox), via ethernet or wifi, in order to access the public Youtube and Orange websites using an ADSL line. Thus can we perform realistic tests, totally representative of the page loadtimes a customer may experience at home.

We also have one PC equiped with a 4G card, enabling access to the public web sites with the 4G network, for evaluating the three protocols in a wireless environment (and its intrinsic and non controllable fluctuant network conditions). This shows the experience mobile end-users can have with their smartphone.

B. Perfy

The *Perfy* tool was developed in our team and allows to script tests, automatically start them, analyse network traffic generated during the page loading and compare the results of the performance metrics supplied by the Navigation Timing API [10] integrated in the Chrome/Chromium browser. This API supplies timing information related to the chained process of page loading.

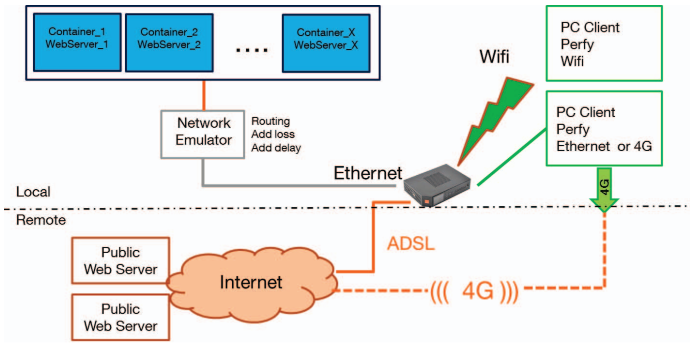

Fig. 2. The testbed

The tool is developed with the Python language, includes the Selenium Webdriver library [12] and launches the Chromium browser (v53) with the defined configuration.

1) Page load time statistics: Perfy is used to launch a browser session and access the website requested. After the page is loaded, it accesses the Navigation Timings and stores the performance metrics in a database. This process is repeated for the number of test iterations requested.

2) Traffic analysis: Perfy also enables the capture of network traffic during the page loading. The resulting captures are then processed and dissected to extract and analyse the different flows (TCP, UDP) and protocols : total number of connections, connections per domain, waterfall charts, etc.

C. Test methodology

In this paper, we mainly focus on the Page Load Time (PLT) [10], which is a metric defined by the W3C as the value for *loadEventEnd* minus *NavigationStart*. The two types of client-server connections in terms of handshake were tested: first ever connection and repeat connection.

1) First ever connection: The webpage is loaded directly in the browser after starting the incognito session. The client connects for the first time to the server and therefore needs to acquire the certificate necessary to pursue a communication between the two. This therefore requires 2 or 1 RTT.

2) Repeat connection: The website is loaded once after starting the browser session. The page is refreshed afterwards in the same session. Between the 2 loadings of the website, the networking service is disabled in order to close the connections still active, and then re-enabled before refreshing. The returned performance metric corresponds to the second page loading (refresh). In this case the client assumes that the server uses the same certificate as the one acquired after the first connection and will directly use this certificate. This therefore requires 1 or 0 RTT.

To provide statistically relevant results for the Page Load Time, each test consisted in 100 iterations for each protocol.

IV. Evaluation & Analysis

This section describes the main results of the tests performed on our local and remote testbeds which are used to identify where the QUIC protocols is the most efficient, when

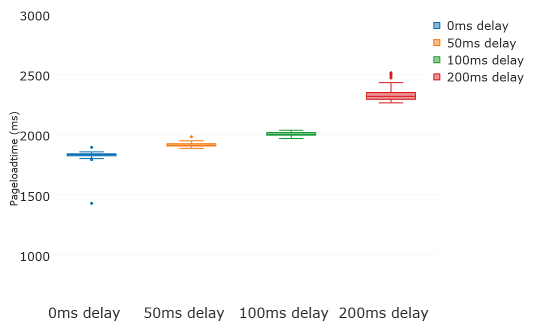

Fig. 3. PLT for QUIC repeat connections for Youtube in case of delay

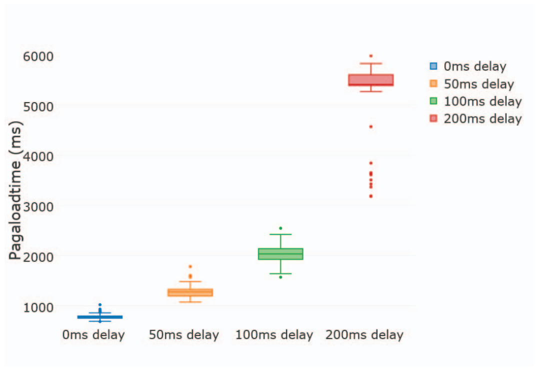

Fig. 4. PLT for HTTP/2 repeat connections for Youtube in case of delay

it can be of interest and how to use it optimally. The figures show the values of the PLTs measured, presented as Plotly boxplots. The lower edge of each box indicates the lower quartile Q1, the upper edge the upper quartile Q3 and the middle line is the median. Here the ends of the whiskers are those of a Tukey boxplot (lowest datum within 1.5 IQR (interquartile range) of Q1 and highest datum within 1.5 IQR of Q3).

A. QUIC wrt Network

In this section, we analyse the QUIC performance, depending on the type of access network and with packet loss and delay generation in the path. The tests were performed on the Youtube website (the public one or the Youtube replica for the local testbed).

1) Behaviour in case of delay: On our local testbed, different values of delay (0ms, 50ms, 100ms and 200ms) were applied, with no loss, and the page load times measured. These values were chosen as representative of real network values [13]. The PLT almost doubles for QUIC first connections but Fig. 3 shows that the impact is not as consequent for QUIC repeat connections as it only increases by 400ms. For HTTP/2 repeat connections, it is much more consequent since the PLT is multiplied by 5 when delay increases (Fig. 4). This showed that QUIC connections are much less sensitive to delay than HTTP/2 connections.

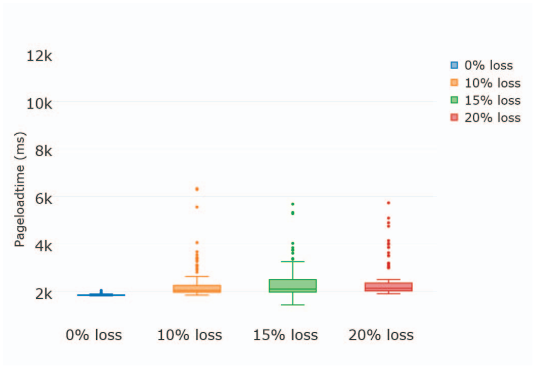

Fig. 5. PLT for QUIC repeat connections for Youtube in case of loss

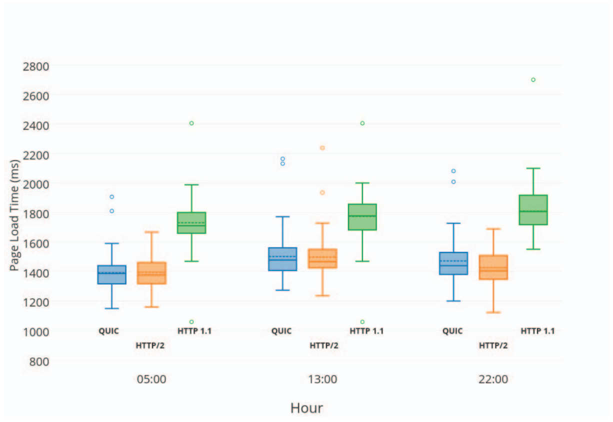

Fig. 7. PLT for a repeat connection on the ADSL link for Youtube

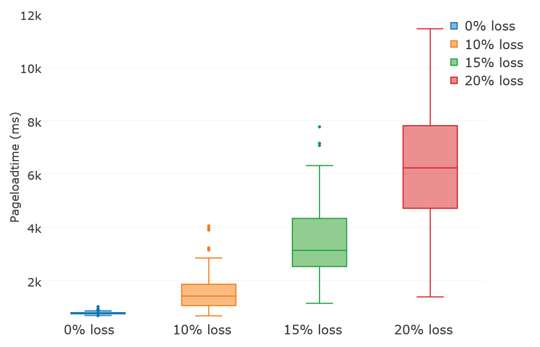

Fig. 6. PLT for HTTP/2 repeat connections for Youtube in case of loss

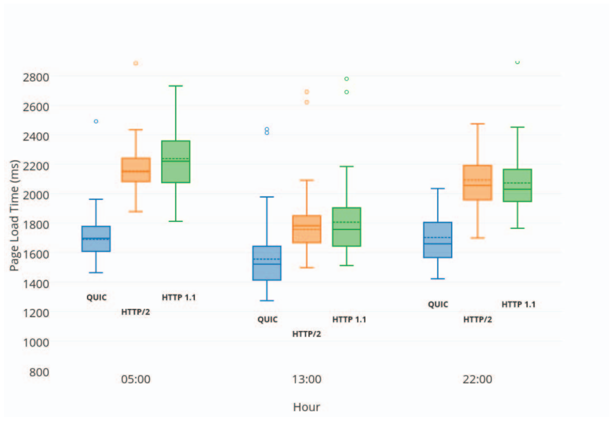

Fig. 8. PLT for a repeat 4G connection for Youtube

2) Behaviour in case of loss: Different values of loss (0%, 10%, 15% and 20%), but no delay, were applied and the page load times measured. Loss values above 10% would mean very high loss in a network. As seen in Fig. 5 and Fig. 6, the test showed that the PLT increased very significantly for QUIC first connections with PLT values going from an average of 2000ms (0% loss) to 16000ms (20% loss), whereas PLT values for QUIC repeat connections went from around 1900ms (0% loss) to 2200ms (20%). Repeat connections are therefore much less sensitive to loss. The PLT values for HTTP/2 repeat connections also increased significantly, going from an average of 800ms (0% loss) to 6500ms (20% loss). The comparison between the two protocols should only take into account the behaviour and not the absolute values of the PLT because the values of PLT for the HTTP/2 protocol on our testbed are intrinsically better than those for QUIC due to the implementation of the server. The tests thus showed that QUIC connections are much less sensitive to loss than HTTP/2 over TCP/TLS connections.

3) 4G vs ADSL: To check our previous conclusion that QUIC can be of interest for wireless networks or similar lossy networks, we performed tests using our remote testbed, enabling us to access remote existing web sites, using public 4G and public ADSL links.Measurements were interleaved for the three protocols (HTTP/1.1, HTTP/2 and QUIC) in order to take place more or less at the same time, and thus ensure a fair comparison. Fig. 7 and Fig. 8 show the results of our tests. For ADSL links, the performances of HTTP/2 and QUIC were similar and better than HTTP/1.1, but for 4G links, the performances of HTTP/2 and HTTP/1.1 were similar and this time QUIC performed better. The most noticeable difference was in low load hours where QUIC PLTs were about 400-500ms better than HTTP/2 & HTTP/1.1 PLTs. The mobile network case is the only one in which we saw QUIC largely outperform HTTP/2.

One of our future works will be to test the 4G case in a realistic mobility model and see how much the varying network conditions and frequent IP connections/disconnections may influence the results.

4) Impact of network load : In this test, we aim to see if the PLT performances varied with the time of day or the day of the tests, depending on the network load, on the real ADSL networks (fixed network, often over dimensioned and with a very small packet loss rate). Here again measurements were interleaved for the three protocols (HTTP/1.1, HTTP/2 and QUIC), to ensure that all protocols experienced the same network conditions. The tests were performed on the Youtube website because it has quite a complex structure.

Fig. 7 shows the results for a weekday, showing better performance on a low load hour. PLT performances measured over a weekend were better, indicating lighter network load. The results depended greatly on the day of the tests. Fig. 7 also shows that the PLT performances of QUIC and HTTP/2 are very similar and are better than those of HTTP/1.1. Our first conclusions are that there is no real benefit of using QUIC at given times of the day or given days of the week. But neither is there obvious evidence to rather use HTTP/2 over TSL+TCP, since results are quite similar.

Based on our evaluation, we can say that the benefits of QUIC for end-users are not so obvious, compared to HTTP/2, regarding the PLT metric. Indeed, HTTP/2 already implements the multiplexing, which largely improves the response time. The main difference is the time to establish the connection (0 or 1 RTT vs 2 or 3 RTTs in case of TCP/TLS handshake). But this time is very small, and not critical for end-users. Furthermore, in our tests, we do not take into consideration the *PUSH* mechanism offered by HTTP/2, which can reduce even more the PLT, sending necessary data before being requested by the browser. This can lead to a more reduced time.

However, we can observe a real benefit of using QUIC in very lossy links. Indeed our tests showed that QUIC outperformed HTTP/2. This is mainly because using UDP instead of TCP avoids the Head-Of-Line blocking issue. QUIC could then be a good candidate for lossy links (Satellite, 4G, 5G).

B. QUIC wrt website structure

1) Website internal structure: For this test, two structures of websites were used on our local testbed :

- Complex Website (*Youtube*)
  This complex website is a replica of the current youtube web site, with many components composing the web page and several of them being fetched from various distributed servers and aggregated on the end-user side.
- Simple Website (*Doctor*)
  This simple website is a replica of the current website of the ANR project named Doctor [14], and has a simple structure since all the files are stored on the same server.

Fig. 3, Fig. 9, Fig. 5 and Fig. 10 show the PLT performances for both the *Youtube* and *Doctor* websites when there is delay in the network or packet loss. We can see that page load time performances are better for a less complex site, but the behaviour in case of delay is the same. This means that website developers should optimise (simplify) their web site structure, so as to deliver contents more quickly.

2) Website distributed architecture: In this test, we aim to compare the PLT from a website having all the contents located on the same server, or the content distributed on several web servers.

On Fig. 11, we can see that the PLT performances are relatively similar for 1, 2 and 4 servers, but decrease for 8 servers. We can thus say that there is no real advantage of distributing the contents on several web servers. The end-user gets the contents more rapidly when they are located on a small number of servers: in this way, we take better advantage of the multiplexing of streams over a single QUIC connection. We observe the same performance result for the first ever connection to the website, as shown in Fig. 12 and Fig. 13. Note that the PLT performances are greatly increased with the –origin-to-force-quic-on flag set in Chromium. When this flag is set, the browser will immediately try to connect to servers by using the QUIC protocol, instead of HTTP/2 over TCP.

To conclude this section, we can say that the design of

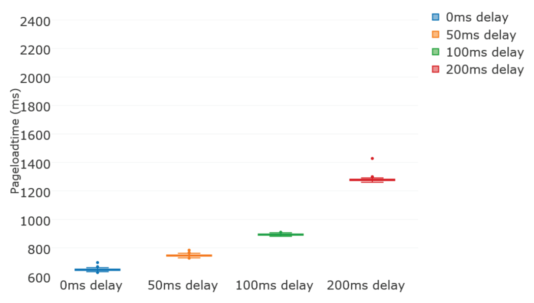

Fig. 9. PLT for repeat QUIC connections for Doctor website in case of delay

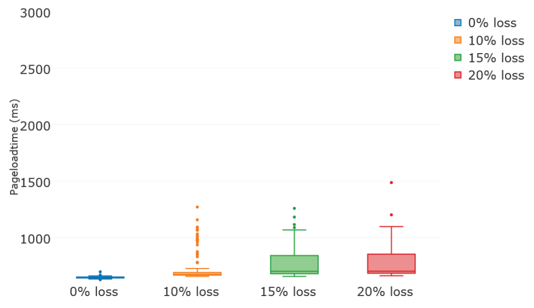

Fig. 10. PLT for repeat QUIC connections for Doctor website in case of loss

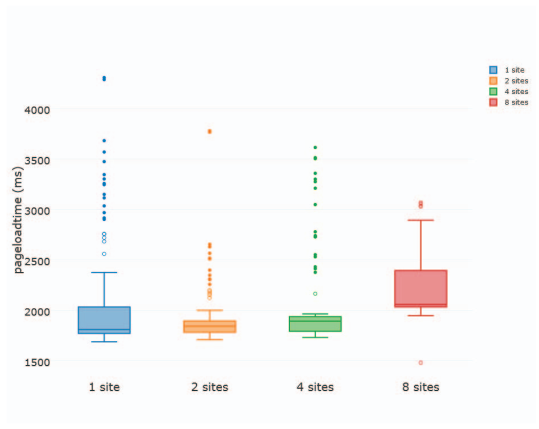

Fig. 11. PLT for repeat QUIC connections for Youtube website with content distributed on 1, 2, 4 and 8 servers

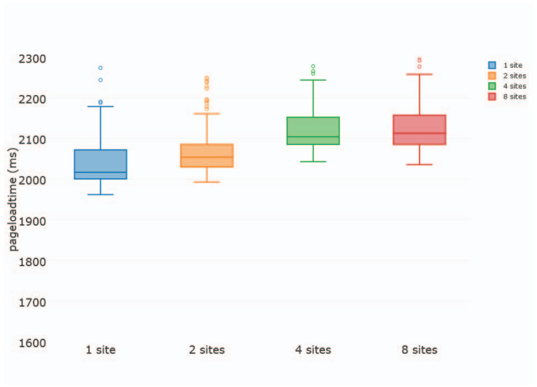

Fig. 12. PLT for first QUIC connections for Youtube website with content distributed on 1, 2, 4 and 8 servers without the flag –origin-to-force-quic-on

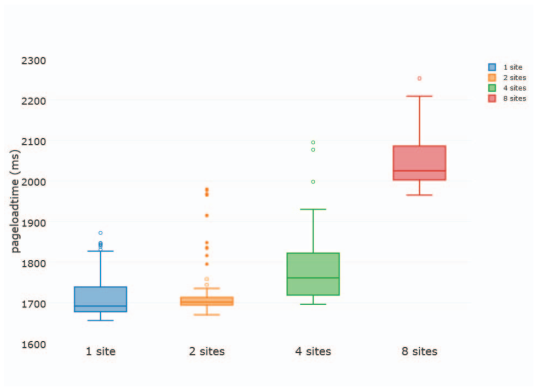

Fig. 13. PLT for first QUIC connections for Youtube website with content distributed on 1, 2, 4 and 8 servers with the flag –origin-to-force-quic-on

QUIC and the preliminary tests we performed show that the adoption of QUIC can lead to a different content placement strategy than today.

Being encrypted, including the headers, QUIC prevents any processing in the network between the end-user and the server. This is then a major concern for middlebox providers (e.g. TCP optimizers, proxy, firewall or parent control boxes, etc.). All actors in the middle of the delivery chain will then have no influence over the QUIC traffic and then lose their activity. It is a critical move for them. Finally, with QUIC, only the browser providers and the server providers will have full knowledge of the end-users browsing, and thus will be able to fully monetize it: e.g. with more ads added in the page, ads under their control, or having knowledge of end-users habits and preferences, etc. Intermediate entities (not acting as QUIC servers) will have no further role.

V. Conclusion

In this paper, we have presented our analysis about QUIC, especially in which conditions QUIC presents interesting performances. We have clearly seen that QUIC outperforms HTTP/2 over TCP/TLS in unstable networks such as wireless mobile networks but in case of stable and reliable networks, the benefits of QUIC are not so obvious. In our future work, we plan to evaluate the QUIC performance with regards to user mobility (when IP connections are torn down and set up very frequently). We also discussed in this paper the possible impact of QUIC on the involved actors, and the evolution its deployment can drive. We have seen that the website developers might make some changes in the way to design the website structure and the way to distribute their contents onto several web servers. Regarding the actors located between the end-users and the servers, their role will change because of the encrypted communications and the need to have certificates. Up to now, only Google uses QUIC for its own services and browsers, but more actors will be needed to have QUIC deployed everywhere. Since the performance gain is not so obvious, some discussions may be needed before a wide adoption. This should happen in the recently created IETF working group [9] where Google pushes QUIC.

Acknowledgement

This work is partially funded by the French National Research Agency (ANR) BottleNet project, No ANR-15-CE25-0013-001. The authors would also like to thank Eliza Bucur, who contributed to the development of the Perfy tool.

References

[1] M. Belshe, R. Peon, and M. Thomson, “Hypertext Transfer Protocol Version 2 (HTTP/2),” Internet Requests for Comments, RFC Editor, RFC 7540, May 2015, http://www.rfc-editor.org/rfc/rfc7540.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7540.txt
[2] R. Peon and H. Ruellan, “HPACK: Header Compression for HTTP/2,” Internet Requests for Comments, RFC Editor, RFC 7541, May 2015.
[3] J. Iyengar, I. Swett, R. Hamilton, and A. Wilk, “QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2,” Internet Engineering Task Force, Tech. Rep. draft-tsvwg-quic-protocol-02, Jan. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-tsvwg-quic-protocol-02
[4] G. Carlucci, L. D. Cicco, and S. Mascolo, “HTTP over UDP: an Experimental Investigation of QUIC,” in *Proceedings of the 30th ACM/SIGAPP Symposium On Applied Computing (SAC)*, April 2015.
[5] S. Ha, I. Rhee, and L. Xu, “CUBIC: a new tcp-friendly high-speed TCP variant.” *ACM SIGOPS Operating Systems Review - Research and developments in the Linux kernel.*, vol. 42, no. 5, pp. 64–74, July 2008.
[6] P. Megyesi, Z. Kramer, and S. Molnar, “How Quick is QUIC?” in *Proceedings of IEEE International Conference on Communications (ICC)*, May 2016.
[7] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru, “How Secure and Quick is QUIC? Provable Security and Performance Analyses,” in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, May 2015.
[8] R. Shade and M. Warres, “HTTP/2 Semantics Using The QUIC Transport Protocol,” IETF Draft, IETF, Draft, July 2016. [Online]. Available: https://datatracker.ietf.org/doc/draft-shade-quic-http2-mapping/
[9] QUIC IETF working group. [Online]. Available: https://datatracker.ietf.org/wg/quic/charter/
[10] Z. Wang, “Navigation Timing,” W3C Recommendation, Tech. Rep., Dec. 2012.
[11] github.com/devsisters. goquic, QUIC support for Go. [Online]. Available: https://github.com/devsisters/goquic
[12] Selenium, browser automation. [Online]. Available: http://www.seleniumhq.org/
[13] I. Grigorik. (2012, jul) Latency: The New Web Performance Bottleneck. [Online]. Available: https://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/
[14] (2016) The ANR Doctor Project. [Online]. Available: http://www.doctor-project.org/