

Google QUIC performance over a public SATCOM access

Ludovic Thomas, Emmanuel Dubois, Kuhn Nicolas, Emmanuel Lochin

▶ **To cite this version:**

Ludovic Thomas, Emmanuel Dubois, Kuhn Nicolas, Emmanuel Lochin. Google QUIC performance over a public SATCOM access. International Journal of Satellite Communications and Networking, 2019, 37 (6), pp.601-611. 10.1002/sat.1301 . hal-02401946

HAL Id: hal-02401946
https://hal.science/hal-02401946

Submitted on 10 Dec 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Open Archive Toulouse Archive Ouverte (OATAO)

OATAO is an open access repository that collects the work of some Toulouse researchers and makes it freely available over the web where possible.

This is an author's version published in: https://oatao.univ-toulouse.fr/21973

Official URL: https://doi.org/10.1002/sat.1301

To cite this version :

Thomas, Ludovic and Emmanuel, Dubois and Nicolas, Kuhn and Lochin, Emmanuel Google QUIC performance over a public SATCOM access. (2019) International Journal of Satellite Communications and Networking, 37 (6). 601-611. ISSN 1542-0973

Any correspondence concerning this service should be sent to the repository administrator: tech-oatao@listes-diff.inp-toulouse.fr

Google QUIC performance over a public SATCOM access

Ludovic Thomas*, Emmanuel Dubois†, Nicolas Kuhn† , Emmanuel Lochin*

* ISAE-SUPAERO, Toulouse, France
† CNES, Toulouse, France

Abstract

Google QUIC accounts for almost 10 % of the Internet traffic and the protocol is not standardized at the IETF yet. We distinguish Google QUIC (GQUIC) and IETF QUIC (IQUIC) since there may be differences between the two. Both Google and IETF versions run over UDP and cannot be split the way satellite systems usually do with TCP connections. The need for adapting any-QUIC parameters needs to be evaluated. Since GQUIC is available, we analyze its behavior over a satellite communication system. In our evaluations, GQUIC quick connection establishment does not compensate an inappropriate congestion control. The resulting page downloading time doubles when using GQUIC as opposed to the performance with optimized split TCP connections. This paper concludes that specific tuning are required when any-QUIC runs over a high BDP network.

1 Introduction

Quick UDP Internet Connections (QUIC) is a transport-layer protocol running on top of User Datagram Protocol (UDP) [17] developed by Google since 2012 and currently under discussion at Internet Engineering Task Force (IETF) [8]. We distinguish Google QUIC (GQUIC) and IETF QUIC (IQUIC). At the time of writing (December 2018), most of the milestones that are related to a first specification of IQUIC (also called QUIC v1) are expected in July 2019 while extansions on *e.g.* multipath IQUIC are expected in 2019 and 2020. Even if IQUIC is not standardized yet, Google has deployed services over QUIC: GQUIC accounts for 2.6 % to 9.1 % of the Internet traffic with rapidly changing versions [23]. To contribute to the specifications of the protocol at IETF, we need to assess the performance of the actually deployed version to justify the need for modifications for high Bandwidth Delay Product (BDP) networks.

Any-QUIC benefits from years of development, rapid deployment and large scale testing. Despite evolving versions, several properties are expected to remain invariant such as encryption of both application data and transport parameters. Fully-encrypted any-QUIC might lead to discrepancies between Over-The-Top (OTT) protocol design choices and Internet Service Provider (ISP) policing mechanisms. While OTT and ISP may not always be seen as competitors [2], the deployment of any-QUIC could lead to some ISP issues: (1) to select the appropriate Quality-of-Service (QoS) policy for the applications carried out; (2) to enable the right shaping policy according to both end user's contracts and access network characteristics; or (3) to optimize the use of the constrained resources such as on cellular networks.

As a matter of fact, operational ISP networks do not evolve at the same pace as End-to-End (E2E) protocols. Furthermore, they should not only be influenced by new emerging protocols, but also with existing and potentially old fashioned protocol stacks. Indeed, both the low Transmission Control Protocol (TCP) Initial congestion Window (IW) values measured in [11] and the analysis of TCP variants in the wild [30] highlight that some web services are still using outdated transport protocol flavours. Any-QUIC could balance the part of old stacks currently used which is a great illustration of the impact OTT have over Internet traffic.

SATellite COMmunication (SATCOM) networks typically break the TCP connections to adapt the transport protocol for long delay links. Although recent E2E protocols may exhibit

decent performance over high BDP paths, splitting TCP allows for adaptation of both TCP slow-start and loss-recovery mechanisms. This results in lower page load times [19]. Moreover, older stacks would anyway need specific acceleration. It is worth pointing out that cellular networks may also introduce the same kind of Performance Enhancing Proxy (PEP) to adapt *e.g.* TCP for the upcoming Fifth-Generation Mobile Communications System (5G): this is not only seen through research papers [10, 13] but also in 3rd Generation Partnership Project (3GPP) study items [22]. In this context, the trend towards the deployment of protocols like any-QUIC questions the actual E2E protocols' adaptations. This motivates this study that assesses the performance obtained by GQUIC when it comes to downloading two different web pages over a geostationnary satellite.

To the best of our knowledge, this paper reports the first evaluations of GQUIC using a real public SATCOM access by assessing the web browsing QoE. Our main findings are:

- for a large web page, the page load time is approximately twice longer with GQUIC compared to Transport Layer Security (TLS)/TCP (section 3.1);
- this difference in larger page load time resides in the poor performance of the non-delegated Congestion Controller (CC) in GQUIC (section 3.2);
- although faster, GQUIC connection establishment does not compensate the above issue (section 3.3).

2 Experiment setup

We exploit a public SATCOM Internet access and repeatedly download two pages with different profiles. We report raw Quality-of-Experience (QoE) metrics, such as Page Load Time (PLT), that represent today's end user experience. We do not pretend to properly assess the QoE of web browsing since it depends a lot on the page that is downloaded. We mainly report trends on some QoE-related web browsing parameters.

Our approach provides a fair comparison between GQUIC and a SATCOM-optimized TCP. Controlled experiments could hardly be envisioned since the operator's ground segment implements specific optimizations and IQUIC available frameworks may not be relevant [12].

For the sake of reproducible science, we have released the scripts that have been used to generate our results [26].

2.1 SATCOM and 4G Internet accesses

To better explain the behavior of GQUIC over a SATCOM access, we also perform some tests with a 4G access as a reference. This section focuses on the description of the SATCOM Internet access.

The public SATCOM operates in Europe with geostationary satellites. We have used a KA-SAT PRO25Go access. To roughly estimate the likely performance of this access and provide an initial sanity check, we have measured that the network is less congested between 2 pm and 4 pm and have decided to run our evaluations at that moment of the day. Early evaluations have showed that TCP is split both at the gateway and at the terminal*. However, we can not have much more information on the stacks that are used within the operator network or at the terminal.

The data plan of our contract limits the variety of performed experiments, but our tests let us assess the estimated SATCOM end-users' QoE. Furthermore, we reckon this also illustrates the impact of OTT protocol design decisions over a SATCOM provider.

A description of a generic SATCOM architecture can be found in [1, 15].

*TCP PEP deployement has been brought to light based on: (a) computations of TCP-level and TLS-level Round Trip Time (RTT); (b) TCP sequence number evolution compared to expected slow start profile and (c) the `traceroute` tool.

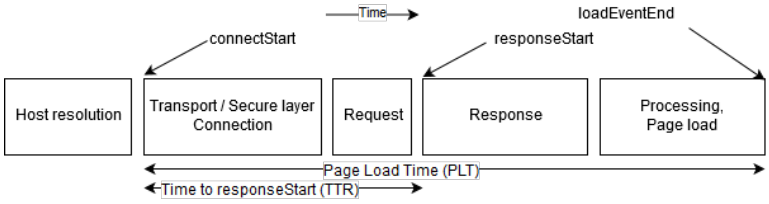


Figure 1: Steps of a page load [28]

2.2 Web pages and QoE

To assess the QoE of a web browsing user, we measure the Page Load Time (PLT), defined as the elapsed time between the `connectStart` and `loadEventEnd` events [28]. For an extensive assessment, other metrics could have been considered such as the time to render or the time to first paint. However, the relevance of these metrics depends on the page that is downloaded and we could not consider many web pages characteristics since the pages needed to be available *via* GQUIC. Our objective is to identify trends and high level comparison of the two protocol stacks.

As shown in Figure 1, the PLT can be decomposed into (1) the time needed to complete the handshake and send the request and (2) the time required to download the content and to process it. In addition to the PLT, we measure the Time To `responseStart` (TTR), the moment at which the user receives the first byte of the Hypertext Transfer Protocol (HTTP) response from the server. This helps us assess the contribution of the connection establishment and the request transmission in PLT. Host resolution is not considered. Metrics are measured when accessing two different targets :

- *Target A :* one picture with a 5.3 MB total size;
- *Target B :* one Google's 404 page with two objects and 11 kB of total size.

Both page are hosted on Google's GQUIC capable servers. We acknowledge that selecting these targets has an impact on the relevance of our conclusions. We do not aim at assessing the performance of GQUIC for any page download but rather expect to compare its behaviour when it comes to downloading large web objects.

2.3 Scenario configuration

Involved protocols collect network and server information using parameters caching, TCP Fast Open (TFO) [6], GQUIC discovery [7] and GQUIC connection resumption[†] mechanisms. The cached parameters are then used to improve the following loads[‡]. To analyze their impact on PLT and TTR, each test unit is composed of three web pages downloads before purging the browser profile. For each load, the client fetches one of the web pages and then closes the browser when the page is retrieved. Elapsed time between two loads is uniformly distributed between 5 and 15 seconds.

Automated weather and link quality reports are linked to the test units. Measures were performed during good weather conditions, with no rain and few clouds. Average link characteristics are presented in Table 1. We also provide worth-noting comparison of the size of the targets with the BDP and the IW in Table 2. We assume servers' default IW is set to 32 TCP Maximum Segment Size (MSS) as observed in GQUIC source code [21]. The actual IW may not be exactly 32, but the main point of this discussion is to highlight the differences between the size of the file and the expected IW. We use the Selenium automation tools to control the browser and retrieve PLT and TTR. Tests are operated on a laptop with 4.15.0-29-generic Linux.

[†]Also called zero-RTT and denoted 0RTT in the following.

[‡]For instance: caching the certificates reduces the handshake duration both with GQUIC and with TCP/TLS; caching the RTT estimate prevents spurious retransmissions during the GQUIC handshake; etc.

Table 1: SATCOM and 4G access networks

Metric	SATCOM	4G
Downlink capacity	25 Mbps	5 Mbps
Uplink capacity	5 Mbps	2.5 Mbps
RTT	750 ms	80 ms

Table 2: Ratios between target sizes, BDP and IW - *e.g.* Target A is 113 times larger than the IW

Target	Size / BDP		Size / IW
	SATCOM	4G	
A (5.3 MB)	4.8	212	113
B (11 kB)	0.01	0.44	0.2

2.4 Few words on the browsers

An analysis of `Chrome`'s behavior combined with [17, 7] has provided us with the following expectations:

- When `Chrome` starts, it opens connections with several servers. If GQUIC is enabled, we have identified that `Chrome` benefits from those loads to open several GQUIC connections. To ensure the same start point whether GQUIC is enabled or not, we have decided to block any GQUIC traffic not intended to the server holding the web-page or its objects. We have done that since we do not want `Chrome`'s specific implementation of `Chromium` to have an impact on the conclusions that are proposed in this article.
- Before using GQUIC, `Chrome` needs to learn about its availability. GQUIC discovery procedure is described in more details in [7, 17]. Important to note here is that `Chrome` always use TCP for the first time it contacts a server.
- Since `Chrome` is restarted between each load, it will neither use GQUIC nor TLS connection resumption. This is due to the way `Chrome` deals with certificates, not to any limitation in the two protocols. We thus expect only 1RTT GQUIC connections.

For each test unit, we use two instances of `Google Chrome 67.0.3396.99`:

- `ChromeQuic`: GQUIC is enabled with Bottleneck Bandwidth and Round-trip propagation time (BBR) congestion control instead of CUBIC[§]. We request BBR on the server side *via* flags during the handshake, but cannot assess the BBR version used by Google servers. However, we can expect that Google had deployed the last version at the time of testing.
- `ChromeNoQuic`: GQUIC is disabled.

`ChromeNoQuic` will always use a HTTP2-TLS1.2-(split)TCP (HTT) stack, whereas `ChromeQuic` starts with HTT and switches to HTTP2-GQUICv39-UDP (HQU) whenever possible. For both instances, TFO is enabled and content caching is disabled. Changes described in this section are the only ones performed on publicly available `Google Chrome`.

3 Results

3.1 PLT for a large page

We denote by HTT the HTTP2-TLS1.2-(split)TCP stack and by HQU the HTTP2-GQUICv39-UDP stack. Figure 2 presents the PLT for target A as a function of the load index since last

[§]This is performed following the method described in [18]. The objective is to get a consistent comparison as BBR is expected to be deployed for TCP on Google's infrastructures [4].

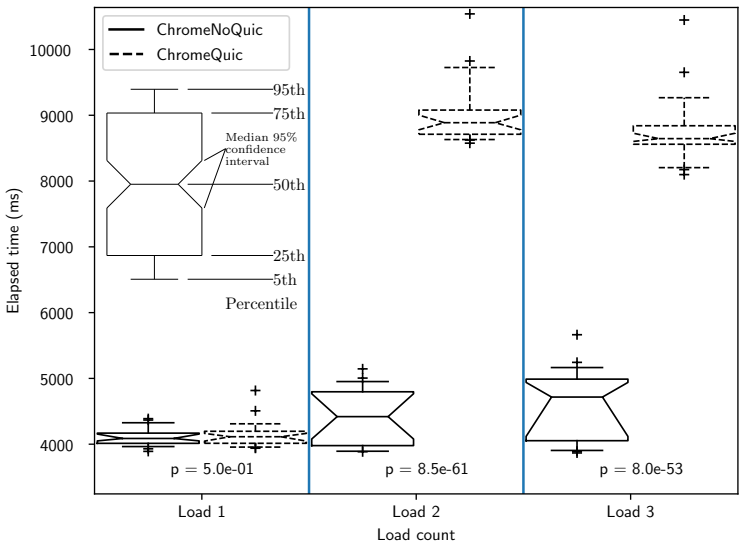


Figure 2: PLT for target A (large picture) on a SATCOM access

profile purge. Metrics are computed according to [3] over 40 test units. At the bottom of the figure, we also provide the *Welch's t-test p* parameter [29] of the two distributions under the *null* hypothesis. For the first load, we expect similar performance for both browser versions as they use the same protocol stack (HTT). The two distributions are indeed located at the same time values and they both show low dispersion. However, with `ChromeQuic` learning about GQUIC availability, HQU is then permanently used for that browser starting from the second load.

We observe the worst performance with HQU where PLT is up to twice longer than `ChromeNoQuic` which uses HTT. The extremely low p values confirm the statistical relevancy of that observation. Each of `ChromeNoQuic` and `ChromeQuic` exhibits a higher PLT dispersion and an intersection of the confidence intervals for loads 2 and 3. It substantiates that above mentioned optimization mechanisms are all performed within one load and we do not expect any further evolution of the PLT with additional loads.

3.2 Packet sequence numbers rate

To understand the performance gap between HQU and HTT, we first focus on the second load of target A and more particularly on the second phase of a PLT, *i.e.*, the time elapsed between the reception of the first response byte and the reception of its last.

To mitigate the issue of encryption keys, we define a sequence number equivalent for GQUIC connections. It is defined as the cumulative sum of the bytes received over the connection, scaled in order to reach the same last value as with TCP. We recognize that the value may present local differences compared to stream offsets embedded in GQUIC packets. Nonetheless, we do believe that global behaviors can be compared.

Figure 3 presents those computations on a subset of HTT and HQU connections. Downward [resp. upward] triangles report the location of TTR [resp. PLT] measurements. We observe that the HQU stack fires the `responseStart` event before HTT. However, the download is completed way after. This can be explained by HTT getting "up-to-speed" and showing a stable and high goodput, while HQU ramps up its transmission rate slowly.

We first focus on HQU. To discriminate the origin of this slow increase between (a) a UDP throughput control from the ISP or (b) the BBR Congestion Controller (CC) itself, we can compute the duration of the BBR *Startup* phase[¶] : $t_s = \ln_2(\mathcal{B}/\mathcal{W}_i)\mathcal{R}$, with $\mathcal{B}$ the BDP of

[¶]Note that BBR *Startup* phase uses the same exponential increase profile as Slow Start or Hystart and only

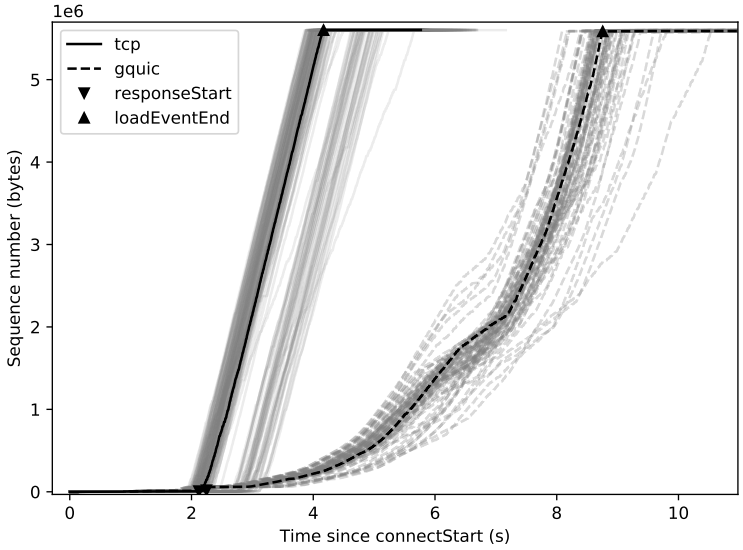


Figure 3: Evolution of the sequence number for the flows downloading the target A on a SAT-COM access

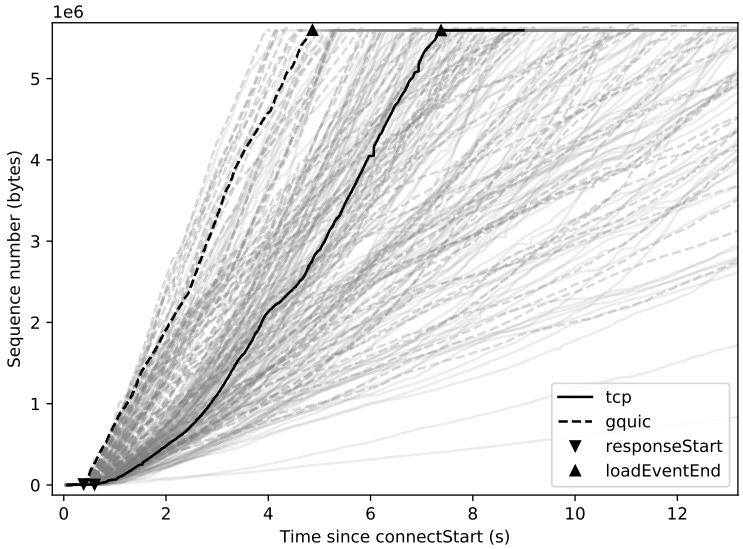


Figure 4: Evolution of the sequence number for the flows downloading the target A on a 4G access network

the link, $\mathcal{R}$ its RTT and $\mathcal{W}_i$ the IW. For the given parameters of the whole E2E link[‖] we obtain $t_s \approx 3.5$s. Compared to expected behavior of BBR we can note on Figure 3 a *Startup* phase of approximatively 4s (between 2s and 6s), followed by three constant-rate segments. The observation is in line with the computed value. With target A only four times bigger than the BDP (Table 2), we can note that the CC above GQUIC spends more than two third of the download in its *Startup* phase. It means that we can expect a similar performance for any other CC using a binary search including TCP New Reno and CUBIC, as long as they are implemented above GQUIC.

In comparison, HTT achieves a near-constant downloading rate, as noted by the quasi-linear increase of sequence numbers. As explained in Section 2.1, the TCP path is split into three connections. One for the satellite link and one at each of its edges: connecting the server to the GateWay (GW) and the Satellite Terminal (ST) to the client (see Figure 5). Let's suppose the segment $PEP_{GW} \rightarrow PEP_{ST}$ uses proprietary protocols and does not require any startup probing phase. The TCP slow start can be neglected for the segment $PEP_{ST} \rightarrow$ Client since its RTT is around 1 ms. Finally, the binary search is also expected to last less than 1 RTT for the segment Server $\rightarrow PEP_{GW}$ before the later toggle down the emission with its flow control[**]. Computations are again in line with the observed values as we note that the final constant TCP throughput is reached in less than 50ms on the row data of Figure 3. In our scenario, splitting TCP and using proprietary protocols in the central segment allows each outer segment to present low BDP and thus permit a fast binary search. On the contrary, GQUIC cannot be split because of transport-level encryption.

To better explain the poor performances of GQUIC, we run the same computations on a comparative 4G access link. Results are shown in Figure 4. First, we note the HQU stack fires the `responseStart` event before HTT and the median gap is around 90ms. Section 3.3 provides insights for that difference. Second, we can note that the BDP of the path is here lower than the IW. Thus, the *Startup* phase is completed in less than 1 RTT. And last, the CCs spend the most part of the load in their *Steady* state because target A is significantly bigger than the BDP (Table 2). On that state, GQUIC shows better performances which is consistent with studies performed on non-split paths [12]. On the geostationary link, the gap in TTR was weak and GQUIC was penalized in its slow start compared to split TCP.

3.3 Focus on the handshake

We just saw that GQUIC encryption prevents any proxy to split the connection, which results in long CC *Startup* phase. But GQUIC was also designed to reduce the handshake duration. In this section, we focus on that phase of the download: from the first packet sent by the client to the first HTTP response bytes received. Results are shown in Figure 6 that presents the notch boxes for the TTR metric.

Loads 2 and 3 show a high dispersion for `ChromeNoQuic` and incompatible confidence intervals for `ChromeQuic`. It questions the hypothesis that learning mechanisms are performed within one load. On a 4G access network (Figure 7), this behavior cannot be seen : we note a gain between load 1 and the followings but loads 2 and 3 show low dispersion and compatible confidence intervals. We assume that the SATCOM ISP policy might disturb the results based on recent traffic history.

One could expect that HQU would gain at least one RTT during the handshake, our results do not reflect that expectation. To understand why, we first need to note that neither GQUIC nor TLS1.2 use connection resumption since the browser is restarted each time. However, analysis of traffic indicates that `Chrome` is using TLS1.2 False Start [16]. The observed HTT handshake is presented in Figure 5. The TCP handshake is performed with the immediate PEP on the path. Its duration can be neglected compared to the rest of the sequence. The TLS1.2 Client Hello

differs in the exit condition.

[‖] Measured throughput : 25Mbps. Measured RTT : 750ms. Default IW : 32.

[**] Throughput before reaching flow control limitation : 25Mbps. Computed RTT based on the difference of the RTT with the Server and with the GW : 30 ms. Default IW : 32.

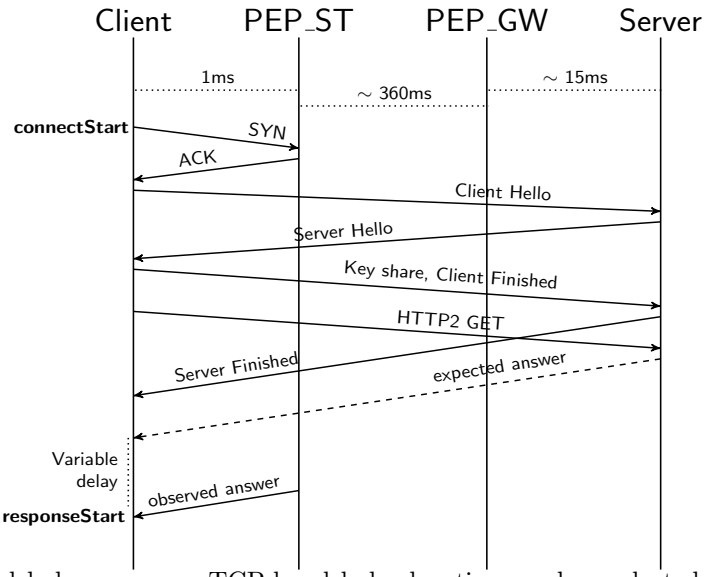


Figure 5: HTT handshake sequence. TCP handshake duration can be neglected thanks to PEP

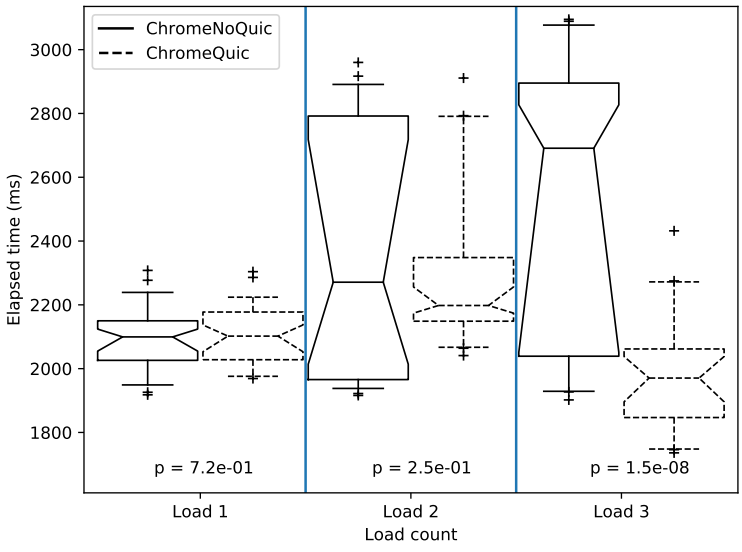


Figure 6: Time to `responseStart` (first HTTP byte) for target A on a SATCOM access

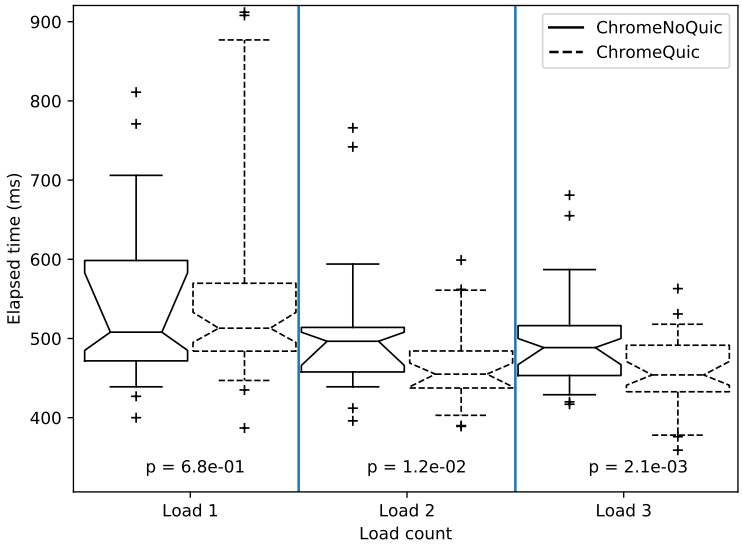


Figure 7: Time to `responseStart` (first HTTP byte) for target A on a 4G access

packet does not suffer any extra delay. So, it appears that the middle segment use a proprietary protocol or already existing connections and no handshake is required between the two PEP. As a consequence we expect to receive the first byte with the HTT stack (*i.e.* TTR_{HTT}) within twice the E2E RTT. In reality, we observe a high-dispersion extra-delay before receiving the HTTP answer (see Figure 5). We put those results in relation with the same dispersive delay observed in [12, 31] and we could not identify with certainty its origin. That being said, the analysis can rely on the median and maximum values of the metric to drive trends.

For the HQU stack, since connection resumption is not used, GQUIC will perform 1 RTT handshakes (Figure 4 of [17]). Here again we can expect : $TTR_{HQU} \approx 2RTT_{E2E}$. In conclusion, thanks to the distributed PEP, to TLS False Start and despite the 1 RTT handshake in GQUIC, both the HTT and the HQU stacks present the same theoretical TTR. In practice, HQU is here slightly faster.

To further justify our analysis, we compare the TTR on a 4G access network (Figure 7). In this network, the TCP handshake cannot be neglected anymore because PEP are not deployed. Thus, $TTR_{HTT} = 3RTT_{E2E}$, *i.e.*, HQU is faster by at least one E2E RTT which is consistent with Figure 4 and Table 1 as the measured TTR gap is around 90 ms. It explains why, when compared to HTT, HQU might present better performances on a mobile network than on a SATCOM access.

3.4 PLT for a small page

In this section, we aim at assessing the impact of the target size on the above mentioned conclusions and further analyze the impact of the ratio between the BDP, the IW. Figure 8 presents those results.

Target B can be sent within an initial congestion window (See Table 2). `ChromeQuic` exhibits a better PLT for loads 2 and 3 of target B than `ChromeNoQuic`. Indeed, the CC does not need to probe the link and the main contributor of the PLT should be the TTR. Moreover, as we saw in section 3.3, HQU generally fires the `responseStart` event slightly before HTT.

That being said, as opposed to target A, objects of target B are located on `www.google.com`[††]. These objects are downloaded by `ChromeQuic` using an already existing GQUIC connection. This is due to the initial connections to `www.google.com` that `Chrome` opens when starting.

[††]Given by analysis of the internal log of `Chrome`

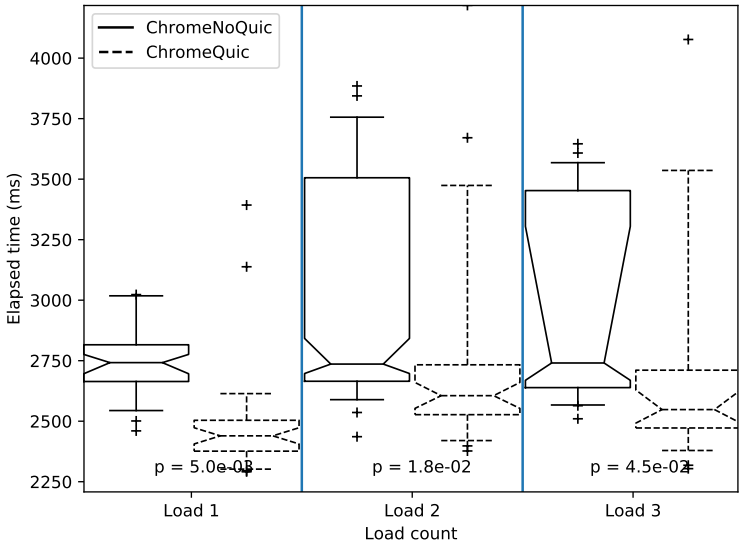


Figure 8: PLT for target B

They enable `ChromeQuic` to discover GQUIC support and even to reuse the previously opened GQUIC connection to fetch the two objects of page B. *I.e.*, by reducing the duration of several handshakes to several servers, the gains are summed up and the PLT is reduced. However, we note that the gain is limited compared to the PLT difference with target A.

4 Contribution and related work

Performance comparisons of GQUIC and TCP have already been conducted under various network conditions and for various applications [12, 5]. For instance, the authors in [31] have performed an evaluation of GQUIC on an emulated platform with scenarios involving SATCOM in LEO and GEO contexts. They concluded that GQUIC outperforms TCP but their testbed did not include any PEP. On the contrary, our study demonstrates that transparent proxying is the cornerstone of better Page Load Time with TCP when compared to GQUIC.

The impossibility for GQUIC to benefit from the PEP technology has already been identified in [12]. Here again, the authors concluded that GQUIC continues to outperform TCP even when the later is split by a proxy. Our tests on a real access highlight the influence of complex PEP deployment schemes on the comparative performance of both protocols.

This paper completes the related work by reporting the first evaluations of GQUIC using a real public SATCOM access. We consider two web-pages: one picture with a 5.3 MB total size and one Google's 404 page with two objects and 11 kB of total size. The main conclusions are the following:

- for a large web page, the page load time is approximately twice longer with GQUIC compared to TCP;
- this difference in larger page load time resides in the poor performance of the non-delegated Congestion Controller in GQUIC;
- although faster, GQUIC connection establishment does not compensate the above issue.

As it is, GQUIC may result in poor end user experience on a high BDP network. However, inducing specific tuning for SATCOM networks in the standards and in actual deployments may be complicated.

5 Discussion

The important variability in applications' requirements makes it hard to define a transport protocol that suits them all. The relevance of application and transport layers protocols depends on both (a) how application data packets are generated and carried out and (b) the network underneath. Taking as example a SATCOM Internet access, the performance of different web applications highly depends on the web pages characteristics [24]. There is no "transport layer silver bullet", *i.e.* a transport protocol that would suit to any application and any network conditions [25].

5.1 ISP point-of-view

For a SATCOM ISP, the deployment of any-QUIC could be seen as interesting:

- we can expect interesting gains for short flows,
- we would not need PEP that are expensive and sometimes complicated to maintain and operate.

However, the performance gains for large files transmissions, the inadequacy of the end-to-end congestion control to SATCOM links and the rapid evolutivity of the protocol may be seen as a threat to good end-user quality of experience.

Moreover, such as illustrated by the discussions on the spin-bit at the IETF, ISP could encounter issues in identifying where the loss occurs in the network.

5.2 Towards a middlebox-friendly any-QUIC

To date, GQUIC does not seem to be blocked by ISP companies, according to [23] and our experiments. It may not be the case when this UDP traffic becomes a greater proportion of the total traffic. Indeed, for the reasons mentioned in 5.1, quickly evolving protocols may not be easy to deal with, from an ISP point-of-view. Interactions between the end-to-end protocol and the operator middle-boxes could be enabled, such as discussed in [14]. Moreover, bits such as the spin-bit could be made available for the ISP operations, such as load-balancing and statistics on the current flows.

To adapt the congestion control and data rate transmission to a specific SATCOM scenario, more important modifications may be required. They may involve changing the advertised receive window, such as in IP-Explicit Rate Notification (IP-ERN) architectures [20], delegate the security to the network operator for better quality of experiences or update the browser [27]. These specifications could be negotiated at initialization such as discussed in the transport draft of IQUIC [9].

References

[1] Toufik Ahmed, Emmanuel Dubois, Jean-Baptiste Dupé, Ramon Ferrùs, Patrick Gélard, and Nicolas Kuhn. Software-defined satellite cloud ran. *International Journal of Satellite Communications and Networking*, 36(1):108–133, 2018.

[2] Angelos Antonopoulos, Chiara Perillo, and Christos Verikoukis. Internet service providers vs. over-the-top companies: Friends or foes? - short talk. volume 44, pages 37–37, New York, NY, USA, January 2017. SIGMETRICS Perform. Eval. Rev., ACM.

[3] Jean-Yves Le Boudec. *Performance Evaluation of Computer and Communication Systems.* EFPL Press, 2011.

[4] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *Queue*, 14(5):50:20–50:53, October 2016.

[5] Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. HTTP over UDP. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. Proceedings of the 30th Annual ACM Symposium on Applied Computing, 2015.

[6] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. TCP Fast Open. RFC 7413, IETF, December 2014.

[7] Ryan Hamilton. Quic discovery. https://www.chromium.org/quic, October 2014.

[8] IETF QUIC Working Group. Quic working group website. https://quicwg.org, sept 2018.

[9] Jana Iyengar and Martin Thomson. Quic: A udp-based multiplexed and secure transport. Work-in-Progress, Internet-Draft, draft-ietf-quic-transport-16, IETF Secretariat, October 2018.

[10] Xiaoxiao Jiang Jae Won Chung, Feng Li. Driving linux tcp congestion control algorithms around the lte network highway. In *NETDEV 2.1*. NETDEV 2.1, 2017.

[11] Jan Rüth, Christian Bormann, Oliver Hohlfeld. On the use of TCP's Initial Congestion Window in IPv4 and by Content Delivery Networks. Presentation to Measurement and Analysis for Protocols Research Group, Internet Research Task Force, IETF 101, London, 2018.

[12] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. Taking a long look at QUIC. *Proceedings of the 2017 Internet Measurement Conference on - IMC*, 2017.

[13] B. H. Kim, D. Calin, and I. Lee. Enhanced split tcp with end-to-end protocol semantics over wireless networks. pages 1–6. 2017 IEEE Wireless Communications and Networking Conference (WCNC), March 2017.

[14] Mirja Kuehlewind and Brian Trammell. Manageability of the quic transport protocol. Work-in-Progress, Internet-Draft, draft-ietf-quic-manageability-03, IETF Secretariat, October 2018.

[15] Nicolas Kuhn and Emmanuel Lochin. Network coding and satellites. Work-in-Progress, Network Coding Internet Research Task Force, draft-irtf-nwcrg-network-coding-satellites-03, IETF, November 2018.

[16] A. Langley, N. Modadugu, and B. Moeller. Transport Layer Security (TLS) False Start. RFC 7918, IETF, August 2016.

[17] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The quic transport protocol: Design and internet-scale deployment. SIGCOMM '17, pages 183–196, New York, NY, USA, 2017. Proceedings of the Conference of the ACM Special Interest Group on Data Communication, ACM.

[18] Dong Mo, Ian Swett, and Nimrod Aviram. Best practice to test congestion control part of quic. Online, jan 2015.

[19] Nicolas Kuhn. MPTCP and BBR performance over Internet satellite paths. Presentation to Internet Congestion Control Research Group, Internet Research Task Force, IETF 100, Singapore, 2017.

[20] Dino Martin Lopez Pacheco, Tuan Tran Thai, Emmanuel Lochin, and Fabrice Arnal. An ip-ern architecture to enable hybrid e2e/ern protocol and application to satellite networking. *Computer Networks*, 56(11):2700 – 2713, 2012.

[21] Google Chromium Projects. Chromium source. Online, sept 2018.

[22] Simone Provvedi. New SID on Study on NR Enhancements for TCP. RP-181061, 3GPP Study Item, 2018.

[23] Jan Rüth, Ingmar Poese, Christoph Dietzel, and Oliver Hohlfeld. A first look at quic in the wild. In Robert Beverly, Georgios Smaragdakis, and Anja Feldmann, editors, *Passive and Active Measurement*, pages 255–268, Cham, 2018. Springer International Publishing.

[24] R. Secchi, A. C. Mohideen, and G. Fairhurst. Performance analysis of next generation web access via satellite. *International Journal of Satellite Communications and Networking*, 36(1):29–43, dec 2016.

[25] Anirudh Sivaraman, Keith Winstein, Suvinay Subramanian, and Hari Balakrishnan. No silver bullet: Extending sdn to the data plane. In *Twelfth ACM Workshop on Hot Topics in Networks (HotNets-XII)*, College Park, MD, November 2013. Twelfth ACM Workshop on Hot Topics in Networks (HotNets-XII).

[26] Ludovic Thomas. Quic user experience assesment - public repo. https://forge.net4sat.org/cnes/quxa-public, nov 2018.

[27] Viasat. Browser. https://www.exede.com/viasat-browser/, last accessed: 08/08/2018.

[28] Zhiheng Wang. Navigation timing. W3C recommendation, W3C, December 2012.

[29] Wikipedia. Welch's t-test. Online, sept 2018.

[30] P. Yang, W. Luo, L. Xu, J. Deogun, and Y. Lu. Tcp congestion avoidance algorithm identification. pages 310–321. 31st International Conference on Distributed Computing Systems, June 2011.

[31] Han Zhang, Tianqi Wang, Yue Tu, Kanglian Zhao, and Wenfeng Li. How quick is quic in satellite networks. In Qilian Liang, Jiasong Mu, Min Jia, Wei Wang, Xuhong Feng, and Baoju Zhang, editors, *Communications, Signal Processing, and Systems*, pages 387–394. Springer Singapore, 2019.