An Early Benchmark of Quality of Experience Between HTTP/2 and HTTP/3 using Lighthouse

Darius Saif*, Chung-Horng Lung†, Ashraf Matrawy‡
Carleton University Department of Systems and Computer Engineering
Email: Dariussaif*,Chlung†,Amatrawy‡@sce.carleton.ca

***Abstract*—Google's QUIC (GQUIC) is an emerging transport protocol designed to reduce HTTP latency. Deployed across its platforms and positioned as an alternative to TCP+TLS, GQUIC is feature rich: offering reliable data transmission and secure communication. It addresses TCP+TLS's (i) Head of Line Blocking (HoLB), (ii) excessive round-trip times on connection establishment, and (iii) entrenchment. Efforts by the IETF are in progress to standardize the next generation of HTTP's (HTTP/3, or H3) delivery, with their own variant of QUIC. While performance benchmarks have been conducted between GQUIC and HTTP/2-over-TCP (H2), no such analysis to our knowledge has taken place between H2 and H3. In addition, past studies rely on Page Load Time as their main, if not only, metric. The purpose of this work is to benchmark the latest draft specification of H3 and dig further into a user's Quality of Experience (QoE) using Lighthouse: an open source (and metric diverse) auditing tool. Our findings show that, for one of H3's early implementations, H3 is consistently worse than H2 in terms of performance.**

***Index Terms*—Benchmarking, QUIC, HTTP/3, Lighthouse**

I. Introduction

QUIC is an emerging transport protocol which has been developed, and rolled out across services, by Google [1]. It is proposed as an alternative to the combination of TCP+TLS and provides features akin to TCP+TLS (such as loss and congestion control, security [2] and Forward Error Correction (FEC) [3]) as well as new ones like stream multiplexing.

The primary motivation for QUIC is to reduce web page latency, thus bolstering a user's Quality of Experience (QoE). QUIC's major advantages over TCP+TLS are (i) eliminating Head-of-Line Blocking (HoLB) through stream multiplexing, and (ii) fewer Round-Trip Times (RTTs) required on connection establishment, thanks to QUIC's cross-layer design. Google researchers have proposed a disruptive approach rather than extensions to TCP most notably because of TCP's entrenchment in networks and Operating Systems (OS). Rather, QUIC is rapidly deployable, as it runs in user space.

The IETF has begun standardizing their own variant of QUIC. This transport has become the backbone of the next generation protocol HTTP/3 (H3) [4]. As such, Google's implementation is now commonly referred to as GQUIC.

Performance analyses have been carried out primarily considering Page Load Time (PLT). The purpose of this letter is extend upon these analyses from the standpoint of providing better visibility on QoE. To do this, use of Lighthouse [5] is proposed. It is an open source auditing tool which provides information rich metrics and an overall aggregate performance score. Lighthouse is able to capture QoE features (like HoLB and prioritization) not possible strictly through a PLT analysis.

Benchmarking was performed on Chrome Canary to an NGINX server with a custom CloudFlare patch for support of H3. While benchmarking on GQUIC [6], [7] found that it is more suitable in networks with high RTT and can achieve higher peak and average throughput compared to TCP, this study on H3 (with the IETF's version of QUIC) did not yield the same result. Explanations to this are offered in this letter and we invite others to reproduce, and confirm, the results.

The rest of this letter is organized as follows: Section II surveys related works in this area. Details of the setup and metrics used are covered in Sections III and IV, respectively. Then, the benchmarking's procedure and results are presented in Section V and VI. Finally, discussion on the results and the letter's conclusions are made in Sections VII and VIII.

II. Related Work

Because of GQUIC and QUIC's infancy, a number of server implementations, in addition to live traffic testing [8], [7], [9], [10], [11], [12], have been considered in the literature. It has also been tested in different network environments [13].

Carlucci *et al.* [6] have considered a number of parameters in their analysis of GQUIC version 21. Goodput, channel utilization, loss ratio, and PLT have all been investigated. This analysis is concerned with comparing CUBIC [14] based congestion control in GQUIC and HTTP/1.1 as well as their respective performance. It was found that GQUIC had higher goodput in under buffered networks, fared better against lossy networks, and reduced PLT. FEC, not enabled by default, noticeably worsens GQUIC's performance.

Cook *et al.* [9] created a scriptable tool to request HTTP/1.1, H2, or H2-over-QUIC pages. With this tool, they investigate PLT of the respective protocols. Go-QUIC[1] was used to power their server; hosting replicas of popular websites. They had found that QUIC fared better in wireless mobile networks, but its gains were not as pronounced in more reliable networks.

Biswal *et al.* [8] used a GQUIC test server included in Chromium. Unlike Cook's work, their web pages were engineered to be of certain sizes and numbers of Document Object Models (DOMs). They concluded that, as the size of objects on a page increased, GQUIC outperformed H2. Conversely, with more objects per page (especially if all small in size), H2 had an edge over GQUIC. The authors noted that this is counter-intuitive because of GQUIC's connection multiplexing advantage and suspect the server to be a limiting factor.

[1] https://github.com/lucas-clemente/quic-go

Wang *et al.* [15] have implemented IETF draft 0 of QUIC in the Linux kernel. They argued that previous works may not have been fair to QUIC's performance, since it operates in user space. Fle transfers were carried out to investigate throughput. For both protocols, CUBIC congestion control was used. It was found that QUIC reached its peak data rate faster than TCP did, QUIC's throughput was higher in lossy environments, TCP and QUIC fairly shared bandwidth, and that TCP had an advantage for large file transfers in environments with low loss rates.

Fairness, video QoE, and proxying were tackled by Kakhki *et al.*'s study on GQUIC [7]. They tested up to version 34 (the latest is 50). GQUIC's code was modified to tune different parameters and also print debug traces, enabling root cause analysis. They found that GQUIC mostly outperformed TCP in desktop and mobile and was unfair to TCP. When either variable network delays or large numbers of small objects were considered, GQUIC performed significantly worse than TCP. For video, GQUIC's gains were seen in high resolution streaming. GQUIC traffic in a proxy yielded mixed results.

III. Experimental Setup

A. Server Side Setup

With VirtualBox, a Ubuntu 18.04.4 Virtual Machine (VM) hosted the web server. It was allocated 4 processors and 6 GB of memory. Cloudflare's *QUIC, HTTP/3, etc.* (QUICHE) project[2] has been leveraged to provide H3 draft 25, plus TLSv1.3, support to an NGINX v1.16 web server. The draft specification of H3 is advertised to clients in the *alt-svc* header for connections to the server. Let's Encrypt [16] was used to generate certificates, required from a trusted authority.

Cloudflare notes that their H3 patch is not officially supported by NGINX. More importantly, the feature is marked as experimental and is subject to limitations. For example, H3's 0-RTT connection establishment is not implemented. Use of OpenLiteSpeed as a web server for this study was also considered, offering similar support and a similar performance disclaimer. NGINX was chosen due to familiarity.

B. Client Side Setup

The Windows machine hosting the VM was used as the client, shown in Figure 1. The client is loaded with Google Chrome Canary: a nightly built version of Chrome which includes various experimental features, including IETF H3 draft support. On startup, Canary can be instructed to support and negotiate H3 draft specification 25 with compliant servers by providing the flags *–enable-quic* and *–quic-version=h3-25*.

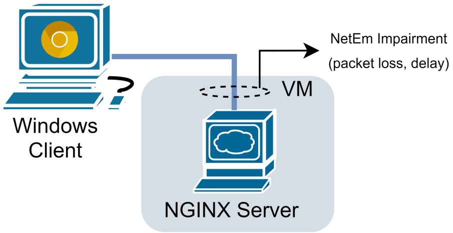

Fig. 1. Network Setup of Experimentation

C. Web Content Served

The web content used in every trial was designed to contain a mixture of content: CSS, JavaScript, text, and images, in order to resemble a modern website. Parameters of interest regarding the web page are presented in Table I:

Total DOMS	85 elements
Max DOM Depth	11 elements
Image Requests	12 (797KB)
Stylesheet Requests	2 (48KB)
Font Requests	1 (31KB)
Document Requests	1 (4KB)
Script Requests	1 (3KB)

TABLE I
Served Web Page Parameters

D. Network Impairments

Modelling various network profiles gives control over the test environment and is critical in benchmarking the respective protocols. As such, NetEm [17], a standard Linux emulation tool, is used. In this study, impairment rules are applied on outgoing packets on the server's network interface. Both packet loss and delays are considered, as shown in Figure 1.

IV. Performance Metrics

Version 5.7.0 of Google's Lighthouse has been leveraged as a tool for collecting QoE performance metrics. Lighthouse is an open source auditing tool included in Google Chrome's DevTools. It measures several characteristics of a web page during its loading and groups them into 5 audit categories. The Performance category is of sole interest for this study. Lighthouse is run locally on a client machine and can be used on any website. The tool prepares a graphical report, which can be downloaded in JSON format, consisting of the metrics and an interactive timeline of page render.

Lighthouse's performance scoring scheme is comprised of three stages: first, raw time values for the metrics are recorded. Then, individual metrics are ranked to a percentile, based on a log normal distribution of sample data from HTTPArchive. To limit outside factors in a web page's performance (network and device variation), a Lighthouse audit engages in CPU and network throttling to normalize sample data. Finally, the individual scores are combined according to a weighting system of each metric's impact on overall performance. The weights assigned to each metric are predetermined and are empirically derived from heuristics.

The combined score, ranging from 0 (lowest) to 100 (highest), ultimately serves as a comprehensive indicator of the user's performance and QoE for a given page. Not only is the percentile ranking system for each metric publicly available, so too is the weighted metric combining scheme[3].

While Lighthouse measures a variety of performance metrics, only 5 are factored into the overall score in version 5.7.0. These metrics, and their weights, are presented in Table II.

[2] https://github.com/cloudflare/quiche

[3] https://github.com/GoogleChrome/lighthouse/blob/master/docs/scoring.md

First Contentful Paint (FCP)	20.0%	The time delta between first navigating to the web page and the browser rendering the very first DOM content
Time to Interactive (TTI)	33.3%	1. The FCP has completed 2. Handlers are loaded for page elements 3. The page responds to input within 50 ms
Speed Index (SI)	26.7%	The time it takes for objects to be visibly displayed during page load
First CPU Idle (FCI)	13.3%	When a page is *minimally* interactive: 1. Most UI elements are interactive 2. The page responds to most user inputs
First Meaningful Paint (FMP)	6.70%	The time it takes for the primary content (the largest above-the-fold layout change) to become visible

TABLE II
LIGHTHOUSE PERFORMANCE METRICS

The developers of Lighthouse, among other experts, maintain that PLT is subjective and loosely defined: arguing that page load does not occur at any *single* instant but is rather a series of milestones. Factors including, but not limited to, HoLB and a page resource prioritization have an impact on what content is populated when and how interactive it is during load. These traits play in to the perceived responsiveness of a web page and are therefore directly tied in to the user's QoE.

The rich collection of metrics in Table II better captures the full picture (request to load and in everything between) than an analysis based purely on PLT, which skips over the user's experience during load. This aspect *should* hold weight when performance testing transport layer protocols.

The scientific dimension of benchmarking that Lighthouse brings to the table is the ranking of its time-based metrics into percentiles, according to the aforementioned distribution. The meaning of raw time data, particularly time deltas between two protocols, can be obscured without (i) a solid baseline expectation on what objectively *good* performance is, (ii) a thorough knowledge of the device(s) and network(s) under test, and (iii) specifics pertaining to the web content served: content type, payload, number of objects, etc. Lighthouse helps address these issues with its dashboard and scoring scheme.

V. PROCEDURE AND METHODOLOGY

Connections were generated through Lighthouse on Chrome Canary to the NGINX server. Only a single connection is made to the server at a time. The protocol (H3 or H2-over-TLSv1.3) was toggled by starting the application with or without the experimental flags. Conveniently, Lighthouse is able to clear the browser's cache before performing an audit, eliminating the potential for a TLSv1.3 0-RTT connection establishment which would make the comparison unfair.

A baseline with no NetEm impairments was captured for both protocols. Then, delay was incrementally introduced to create a higher RTT. At a fixed amount of delay, packet loss was then introduced and gradually increased. For each iteration, a total of 5 audits were performed and packet captures were taken in Wireshark. The raw Lighthouse metric data was averaged in order to deal with any variation. The averaged raw metrics were then translated to an overall Lighthouse score, using its the publicly available scoring calculator.

VI. RESULTS

A. Effects of Delay

Starting from no NetEm impairments, the delay was increased. The overall Lighthouse score for each protocol is shown in Figure 2. At the beginning, the performance of both protocols are quite similar, but as the delay increases page delivery is more severely degraded for H3. After about 500ms, the highest rate of performance degradation is observed.

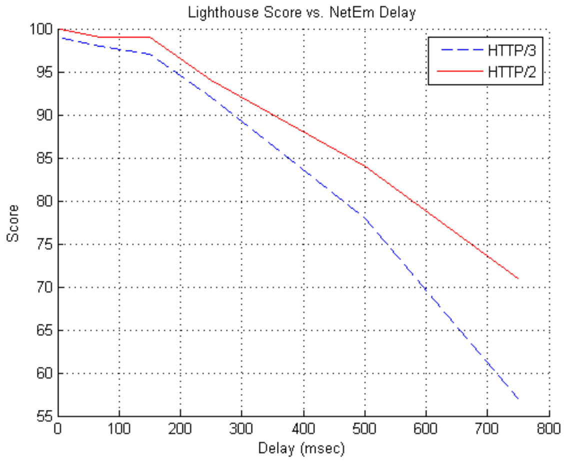

Fig. 2. Effects of Delay on Overall Lighthouse Score

In studies related to GQUIC [7], [9], [15], it had also been noted that with no impairments, TCP based delivery had a slight edge – one possible explanation for this is additional overhead introduced by GQUIC operating in user space rather than the kernel. However, unlike the results shown here, QUIC variants outperformed TCP+TLS as delay increased.

B. Effects of Packet Loss

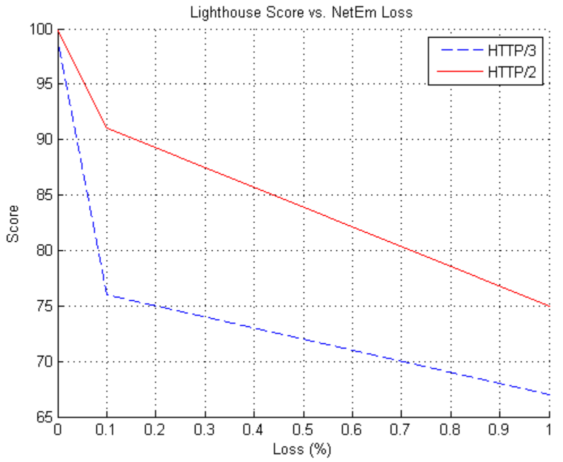

Fig. 3. Effects of Packet Loss on Overall Lighthouse Score

In this experiment, a fixed delay of 250 ms is applied and packet loss is introduced with NetEm. Again, the results in Figure 3 show that H3 trails its predecessor. The Wireshark captures for these experiments show that, with H3, almost twice as packets were sent. The total aggregate bytes did not differ significantly.

VII. Discussion

The early results presented above may seem discouraging. Studies looking at GQUIC have identified scenarios in which that protocol has quite an edge over H2. The question is: why do the general trends of this study, on H3, differ from the community's conclusions of GQUIC? We offer some early explanations about this and invite further studies on H3:

1) Use of Lighthouse Performance Metrics: An identifiable difference between this study and those prior is the use of Lighthouse. With advanced stream multiplexing, addressing HoLB, it was expected that the use of these metrics would actually be more favorable to H3. Alas, all metrics factored into the score were consistently worse for H3. It is not believed that Lighthouse attributed to this difference.

2) Differences with GQUIC and IETF QUIC: GQUIC and IETF QUIC are not the same protocol – in fact, their specifications alone contain innumerable differences. One of which could be GQUIC's use of BBR [18].

3) Limitations of Server Implementations: It should be stressed that implementations of H3 do not claim to be suitable for production environments at this point. They are mainly made available for test purposes. Chunks of the specification are either (i) incomplete (i.e. 0-RTT in QUICHE), or (ii) under test and subject to tuning and bug fixing. The tools available for testing H3 are also sparse and change quickly. During the course of this experimentation, a number of updated draft specifications to H3 had been released. Each time, Canary had deprecated support for the previous draft version, making the client and server unable to speak with each other until QUICHE also added support for that version.

4) Use of TCP+TLSv1.3 with HTTP/2: In this study, H3 is benchmarked against H2+TLSv1.3, the newest version of TLS. Just like QUIC variants, TCP+TLSv1.3 boasts a connection establishment of *at most* 1-RTT. Its predecessor, TCP+TLSv1.2, requires 3-RTTs. Previous studies did not incorporate the new version of TLS into their test environment, which would have given QUIC a performance edge of up to 3-RTTs.

VIII. Conclusions

GQUIC is a low latency transport protocol and alternative to TCP+TLS. Its design has a more disruptive approach due to the entrenchment of TCP in networks and OSs. Furthermore, its cross-layer design and advanced features are able to address multiple TCP+TLS inefficiencies. Its adoption by the suite of Google services has sparked academic research and testing of the protocol. The general consensus is that GQUIC is able to perform decisively better in environments with high RTT and/or packet loss and for pages containing large objects. As such, the IETF has modeled next generation protocols (H3 and TLSv1.3) around these concepts.

Previous works have judged the performance of GQUIC compared to H2 over TCP+TLS. In doing so, a number of server implementations and methods of traffic generation have been considered. The main, if not only, metric employed in the aforementioned works is PLT. Alone, PLT provides little insight into a user's QoE. Rather, this analysis leverages Lighthouse, which utilizes diverse metrics to depict various milestones throughout the page loading process.

As of yet, benchmarking analyses, scrutinizing the performance of H3 draft specifications, have not been exercised. Of course, it is acknowledged that at this point that the specifications are merely drafts. Server implementations and client support available are sparse and are listed as experimental. Given that, the benchmarking conducted showed that H3 fared consistently worse than its predecessor – a number of potential explanations to why this is has also been provided. The intent of this letter has been to open dialogue about H3's performance and the need for metric diversity.

References

[1] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, *et al.*, "The quic transport protocol: Design and internet-scale deployment," in *Proc. the Conference of the ACM Special Interest Group on Data Communication*, pp. 183–196, 2017.

[2] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru, "How secure and quick is quic? provable security and performance analyses," in *Proc. IEEE Symposium on Security and Privacy*, pp. 214–231, 2015.

[3] P. Qian, N. Wang, and R. Tafazolli, "Achieving robust mobile web content delivery performance based on multiple coordinated quic connections," *IEEE Access*, vol. 6, pp. 11313–11328, 2018.

[4] M. Bishop *et al.*, "Hypertext transfer protocol version 3 (http/3)," *Internet Engineering Task Force, Internet-Draft ietf-quic-http-27*, 2020.

[5] "Github: Google chrome - lighthouse." https://github.com/GoogleChrome/lighthouse. Accessed: 2020-01-20.

[6] G. Carlucci, L. De Cicco, and S. Mascolo, "Http over udp: an experimental investigation of quic," in *Proc. the 30th Annual ACM Symposium on Applied Computing*, pp. 609–614, 2015.

[7] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a long look at quic: an approach for rigorous evaluation of rapidly evolving transport protocols," vol. 62, pp. 86–94, ACM, 2019.

[8] P. Biswal and O. Gnawali, "Does quic make the web faster?," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2016.

[9] S. Cook, B. Mathieu, P. Truong, and I. Hamchaoui, "Quic: Better for what and for whom?," in *Proc. IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2017.

[10] Y. Yu, M. Xu, and Y. Yang, "When quic meets tcp: An experimental study," in *Proc. 36th International Performance Computing and Communications Conference (IPCCC)*, pp. 1–8, IEEE, 2017.

[11] P. Megyesi *et al.*, "How quick is quic?," in *Proc. International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2016.

[12] P. K. Kharat, A. Rege, *et al.*, "Quic protocol performance in wireless networks," in *Proc. International Conference on Communication and Signal Processing (ICCSP)*, pp. 0472–0476, IEEE, 2018.

[13] Y. Wang, K. Zhao, W. Li, J. Fraire, Z. Sun, and Y. Fang, "Performance evaluation of quic with bbr in satellite internet," in *Proc. the 6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, pp. 195–199, IEEE, 2018.

[14] S. Ha *et al.*, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[15] P. Wang, C. Bianco, J. Riihijärvi, and M. Petrova, "Implementation and performance evaluation of the quic protocol in linux kernel," in *Proc. the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 227–234, 2018.

[16] "Let's encrypt – free ssl/tls certificates." https://letsencrypt.org. Accessed: 2020-01-20.

[17] S. Hemminger, "Network emulation with netem," in *Proc. Linux Conference Australia, Canberra, Australia, April 2005*, 2005.

[18] N. Cardwell, Y. Cheng, C. S. Gunn, and all, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, no. 5, p. 50, 2016.