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 letter is to benchmark the latest draft specification of H3 and dig into a user's Quality of Experience (QoE) by using Lighthouse: an open source (and metric diverse) auditing tool. Our findings show that, for one of H3's early implementations, H3 is mostly worse but achieves a higher average throughput.**

***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]. Its features akin to TCP+TLS (such as loss and congestion control, security [2] and Forward Error Correction (FEC) [3]) position QUIC as an alternative to the former two. QUIC also brings advanced features like stream multiplexing to the table.

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 comparisons between (G)QUIC and TCP+TLS have primarily considered Page Load Time (PLT). This letter's purpose is to extend upon those analyses from the standpoint of providing better visibility on QoE. As such, use of Lighthouse [5] is proposed. It is an open source auditing tool which provides information-rich metrics and an aggregate performance score. Lighthouse is able to capture QoE features (like HoLB and prioritization) which a PLT analysis cannot.

This letter provides four main points of contribution: (i) an early look at H3's performance, which has not been discussed in literature to our knowledge, (ii) comparison against HTTP/2 (H2) over TCP+TLSv1.3, which is more competitive than TCP+TLSv1.2 in terms of connection establishment, (iii) a discussion on how the differences between GQUIC and IETF QUIC may affect their respective performance, and (iv) incorporating more metric diversity into test scenarios to better represent QoE implications, not widely considered before.

Studies on GQUIC [6], [7] found it to be more suitable than H2-over-TCP+TLSv1.2 in networks with high RTT. Our study between H3 (with IETF QUIC, hereby simply called QUIC) and H2-over-TCP+TLSv1.3 did not yield the same observation. Explanations to this are offered in this letter and we invite others to reproduce, and confirm, the results. Our benchmarking was performed on Chrome Canary to an NGINX server with a custom CloudFlare patch to support H3.

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 methodology 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 (G)QUIC's infancy, a number of server implementations, in addition to live traffic testing [7], [8], [9], [10], [11], [12], have been considered in the literature.

Carlucci *et al.* [6] considered goodput, channel utilization, loss ratio, and PLT in their analysis of GQUIC v21 and HTTP/1.1. Both used congestion control from [13]. They found GQUIC had higher goodput in under-buffered networks, fared better in lossy networks, and reduced PLT. FEC, not enabled by default, noticeably worsened GQUIC's performance.

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

Biswal *et al.* [8] used Chromium's GQUIC v23 server. Unlike [9], their 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 small objects per

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

page, H2 fared better. This was noted as counter-intuitive due to GQUIC's theoretical edge by means of stream multiplexing.

Fairness, video QoE, and proxying were tackled by Kakhki *et al.*'s [7] study on GQUIC versions up to v34. They modified GQUIC's code to tune parameters and also print debug traces, enabling root cause analysis. They found that GQUIC was unfair to TCP+TLSv1.2 and mostly outperformed it on desktop and mobile. When either variable network delays or large numbers of small objects were considered, GQUIC performed significantly worse than TCP+TLSv1.2.

A similar argument against PLT was made in [14], [15]. TCP was closely tuned to GQUIC v43 and human observers rated their QoE. Their PLT alternatives found a slight edge for GQUIC but users weren't able to distinguish either protocol.

III. Experimental Setup

A. Server Side Setup

A Ubuntu 18.04.4 (kernel 4.15.0-88) Virtual Machine (VM) in VirtualBox hosted the web server. It was allocated 4 processors and 6 GB of memory. Cloudflare's *QUIC, HTTP/3, etc.* (QUICHE) project[2] was leveraged (up to commit *98757ca*) to provide H3 draft 27, and TLSv1.3, support to an NGINX v1.16 web server. Let's Encrypt [16] was used to generate trusted certificates, as QUIC does not accept self signed certificates.

H3 support was advertised to clients in the *alt-svc* header for HTTP connections to the server. Both H3 and H2-over-TCP+TLS connections employed TLSv1.3 handshaking and used CUBIC congestion control. Stock TCP tuning was used.

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, at the time of writing, H3's 0-RTT connection establishment was not implemented. Use of OpenLiteSpeed as a web server for was also considered, which offered a similar support and performance disclaimer. NGINX was chosen due to familiarity.

B. Client Side Setup

The Windows 10 machine hosting the VM was used as the client, shown in Figure 1. A speed test on the client yielded a ping time of 20ms, downlink of 52.95Mbps, and uplink of 7.83Mbps. The client was loaded with Google Chrome Canary: a nightly built version of Chrome with various experimental features, including IETF H3 draft support. On startup, Canary can be instructed to support and negotiate H3 draft specification 27 with compliant servers by providing the flags *–enable-quic* and *–quic-version=h3-27*.

C. Network Impairments

NetEm [17], a standard Linux emulation tool, was used to control different network parameters, which was critical in benchmarking the respective protocols. In this study, impairment rules were applied on outgoing packets on the server's network interface. Both packet loss and delay were considered, as shown in Figure 1. Other performance analyses [3], [7], [8], [9], [11] had also used NetEm to this effect.

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

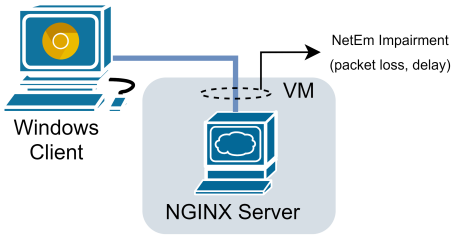

Fig. 1. Network Setup of Experimentation

D. 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 realistic modern website. The web page's parameters of interest 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

IV. Performance Metrics

Version 6.0.0 of Google's Lighthouse was 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 (while the page loads) and groups them into 5 audit categories. The Performance category was of sole interest for this letter. Lighthouse runs locally on a client machine and can be used on any website. The tool prepares a downloadalbe JSON report consisting of the recorded metric data and an interactive timeline of how the page rendered, shown in Figure 2.

Lighthouse's performance scoring scheme is comprised of three stages: first, raw 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 by Lighthouse through 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 6 are factored into the overall score in version 6.0.0. These metrics, and their weights, are presented in Table II.

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

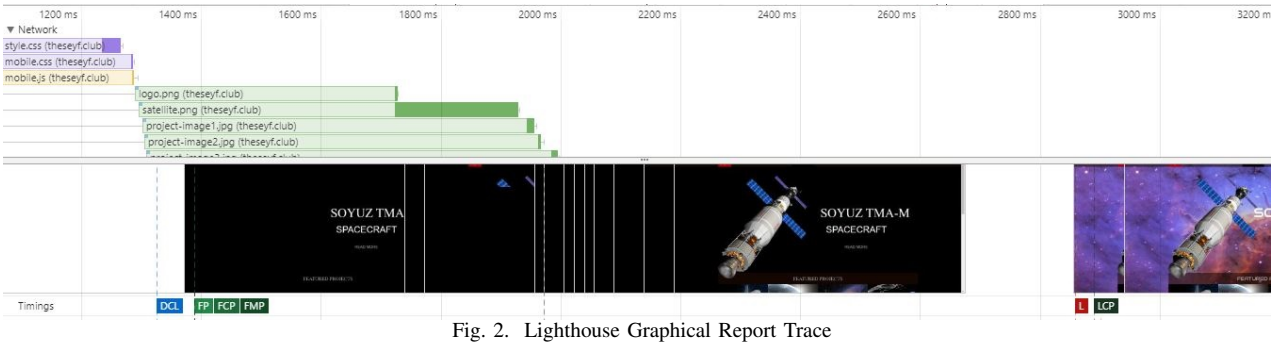

Fig. 2. Lighthouse Graphical Report Trace

First Contentful Paint (FCP)	15%	The time delta between first navigating to the web page and the browser rendering the very first DOM content.
Time to Interactive (TTI)	15%	1. The FCP has completed 2. Handlers are loaded for page elements 3. The page responds to input within 50ms.
Speed Index (SI)	15%	The time it takes for objects to be visibly displayed during page load.
Largest Contentful Paint (LCP)	25%	The time it takes for the element on the page with the largest payload to have been completely rendered.
Total Blocking Time (TBT)	25%	In the time between FCP and TTI, tasks taking longer than 50ms are summed into TBT. Timing starts after 50ms of task execution.
Cumulative Layout Shift (CLS)	5%	Quantifies the page's stability as resources are loaded or DOMs are added. A higher score means more frequent layout shifts.

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 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 captures the full picture (request to load and everything in between) better than an analysis based purely on PLT, which skips over the user's experience during load. A similar observation is made in [14], [15], though metric combining was not covered in their work.

The meaning of raw data, particularly time deltas between two protocols, can be obscured without (i) a solid expectation on what objectively *good* performance is, (ii) 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 percentile based scoring scheme.

V. PROCEDURE AND METHODOLOGY

Connections were generated through Lighthouse on Chrome Canary to the NGINX server. Only a single connection was made to the server at a time. The protocol (H3 or H2-over-TCP+TLSv1.3) was toggled by starting Chrome Canary with or without the experimental flags noted in III.

The browser's cache was cleared before performing every audit, eliminating the potential for either protocol's connection resumption to kick in. This helped ensure the benchmarking's fairness. A baseline with no NetEm impairment 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 aggregate Lighthouse score, using the publicly available scoring calculator.

VI. RESULTS

A. Baseline Measurement

With no impairment from NetEm, a baseline was collected for each protocol. A histogram of the averaged raw metric data is provided in Figure 3 – the lower the value, the better. H3's SI beat its predecessor's but in terms of LCP, H3 fared decisively worse. The unitless aggregate Lighthouse scores for H3 and TCP+TLSv1.3's baseline were 65 and 87, respectively.

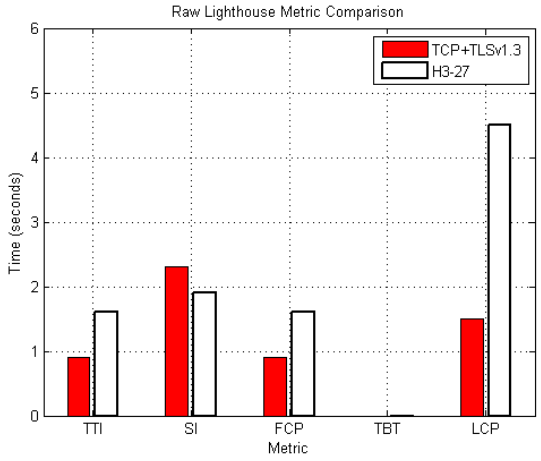

Fig. 3. Baseline Comparison of Raw Metrics

CLS is not shown above as it's not measured in time – it was 0.003 for both protocols. For all the conducted experiments, it was noted that the reported TTI and FCP were the same (that is, $TTI_{H3} = FCP_{H3}$ and similarly for TCP+TLSv1.3), making TBT always 0ms. The line width of H3's bar graph makes TBT appear non-zero.

B. Effects of Delay

Starting from no NetEm impairment, the delay was increased. Figure 4 shows the aggregate Lighthouse score and raw metrics for LCP and SI. These metrics were chosen to be highlighted as LCP is one of the highest weighted metrics and since H3 was noted to have a competitive SI. H3's aggregate score consistently trailed, and the largest score differentials (of 22 and 25) occurred with no impairment and at 300ms.

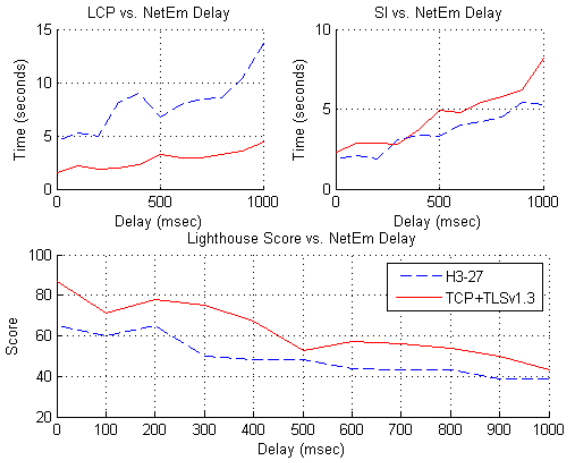

Fig. 4. Effects of Delay on Lighthouse Metrics & Score

In studies related to GQUIC [7], [9], [18], it had also been noted that with no impairment, TCP based delivery had an edge – it was suspected that GQUIC introduced additional overhead by operating in user space rather than the kernel. The same holds true for H3. Unlike studies on GQUIC and TCP+TLSv1.2 however, H3 never overtook its competitor.

The performance gap became quite small approaching 1000ms of delay. TCP+TLSv1.3 still performed better in a trial with 2000ms delay. Although LCP was consistently much worse with H3, its SI remained competitive for the duration of testing. H3's TTI and FCP were consistently worse.

C. Effects of Packet Loss

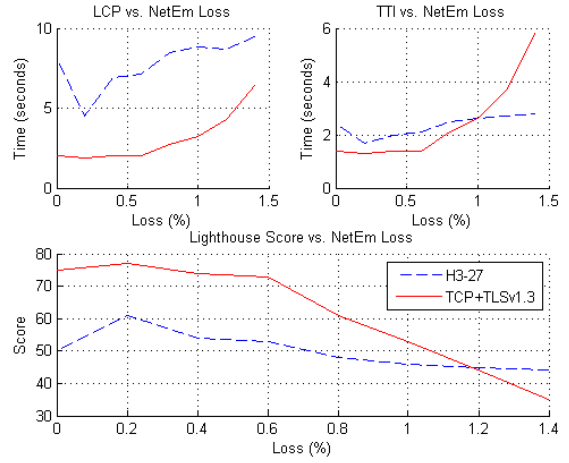

Fig. 5. Effects of Packet Loss on Lighthouse Metrics & Score

For the packet loss test, a fixed delay of 300ms and an increasing loss percentage were introduced with NetEm. Figure 5 shows that, at higher loss rates, H3 overtook TCP+TLSv1.3. H3's aggregate score flattened out as more loss was introduced whereas TCP+TLSv1.3's curve decayed almost linearly. Beyond the setting of 1.4% packet loss in NetEm, the results became quite unstable (in some cases Lighthouse was not able to properly complete its audit) and are hence not included.

Again, H3's LCP was much worse. However, H3's more stable TTI (and FCP) attributed to its higher scoring. The SI values between H3 and TCP+TLSv1.3 were very similar to one another in these trials. Packet captures in Wireshark showed that, with H3, almost twice as many packets were sent. The total aggregate bytes did not differ significantly however.

D. Throughput

In this test, neither Lighthouse metrics nor NetEm impairment were used. A 25MB file download from the server was completed for both protocols, while capturing in Wireshark. Though the plots are superimposed, each file download occurred separately. The throughput results in Figure 6 show that the file download finished 4 seconds faster in H3's favor.

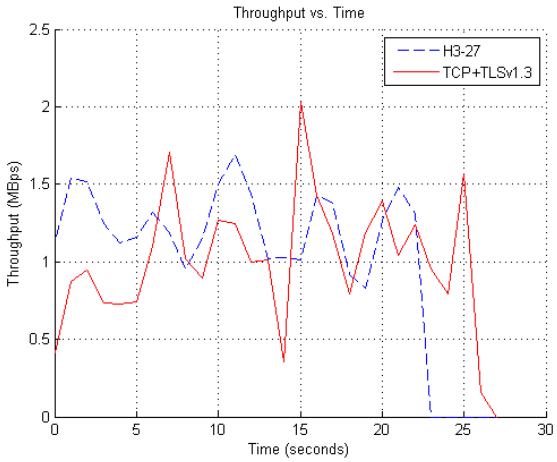

Fig. 6. 25MB File Download Throughput

Be that as it may, TCP+TLSv1.3 achieved a higher peak throughput (2.03MBps) than H3 (1.69MBps). The average throughput for H3 (1.24MBps) was more favorable than TCP+TLSv1.3 (1.03MBps). H3 produced 3% more data on the wire, as it generated more packets than the stock TCP tuning. The fact that H3 finished faster made for an interesting comparison between the delay results presented in Figure 4 and the throughput results in Figure 6.

CloudFlare's CUBIC parameterization used more aggressive *rwnd* and *cwnd* values than that of TCP's stock tuning. These points certainly gave H3 somewhat of a head-start and kept its rate bounded between 1-1.5MBps. This advantage, however, didn't necessarily translate to a better Lighthouse score for the approximately 1MB web page. The larger buffers may have attributed to H3's generally better SI (weighted 15%) scoring but its LCP (weighted 25%) timings were more damning.

VII. Discussion

Studies [6], [7], [8], [9], [10], [11], [18] on (G)QUIC had identified scenarios which it had quite an edge over TCP+TLSv1.2. In comparing H3 to H2-over-TCP+TLSv1.3, the benefits were seldomly apparent and rather marginal. We offer some early explanations as to why this may have been and invite further studies on H3:

1) Lighthouse: With stream multiplexing that addresses HoLB, it was expected that these metrics, and thus QoE, would favor H3 – alas, the aggregate scores were mostly worse. LCP largely attributed to H3's poorer performance but metrics like SI, TTI, and FCP led to more interesting outcomes. Lighthouse was not believed to have tipped the scales towards H2.

2) Differences with GQUIC and QUIC: These two are not the same protocol – in fact, their state machines, source coding, and header framing contain innumerable differences. Notable examples include: (i) more fields in QUIC are encrypted, (ii) QUIC's method of header compression is different, (iii) GQUIC uses a proprietary security scheme – GQUIC Crypto, and (iv) GQUIC uses BBR [19] congestion control. These may, quite feasibly, favor GQUIC's performance.

3) Limitations of Server Implementations: It is stressed that implementations of H3 are made available for test purposes and do not claim to be suitable for production environments at this point. Chunks of the specification are either incomplete or subject to tuning and bug fixing. H3 servers evolve quickly, just as the IETF drafts do. During the course of this experimentation, a number of updated drafts to H3 had been released, prompting a plethora of code churn in server implementations.

Recently, Cloudflare published a blog post[4] with initial testing of their own. It was found that for realistic pages, H3 was 1-4% slower than H2. Although it is not clear if network impairment was considered, their results seem more or less consistent with the results presented in this letter.

4) H2-over-TCP+TLSv1.3: In this letter, H3 was benchmarked against H2-over-TCP+TLSv1.3, the latest version of TLS. Just like (G)QUIC, TCP+TLSv1.3 boasts a connection establishment of *at most* 1-RTT (if TCP Fast Open [20] is used). Its predecessor, TCP+TLSv1.2, required 3-RTTs. Previous studies did not incorporate TLSv1.3 into their test environment, giving QUIC a performance edge of up to 3-RTTs. This made QUIC more desirable in high RTT networks.

VIII. Conclusions

GQUIC is a low latency alternative to TCP+TLS. Its disruptive design approach is due to the entrenchment of TCP in networks and OSs. Furthermore, its features and cross-layer design are able to address multiple TCP+TLS inefficiencies. Following deployment, and academic testing, of the protocol, the general consensus was that GQUIC was able to perform decisively better in environments with high RTT and/or packet loss as well as pages containing large objects. The IETF has modeled the next generation of HTTP around these concepts.

[4]https://blog.cloudflare.com/http-3-vs-http-2/

The main, if not only, metric employed in most of the past works was PLT. Alone, PLT provides little insight into a user's QoE. Rather, this analysis leveraged Lighthouse, which utilized diverse metrics to depict various milestones throughout the page loading process.

Until now, academic performance benchmarking between H3 and TCP+TLSv1.3 has not been presented. Of course, it is acknowledged that at this point, the IETF specifications are merely drafts. Plus, server implementations and client support were sparse and listed as experimental. Given that, our results showed that H3 mostly fared worse than its predecessor: H3 performed better under high loss and achieved a higher average throughput. Discussions and explanations as to why this may have been the case have also been provided.

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-25*, 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] 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.

[14] K. Wolsing, J. Rüth, and all, "A Performance Perspective on Web Optimized Protocol Stacks: TCP+ TLS+ HTTP/2 vs. QUIC," in *Proc. of the Applied Networking Research Workshop*, pp. 1–7, 2019.

[15] J. Rüth, K. Wolsing, and all, "Perceiving quic: do users notice or even care?," in *Proc. of the 15th International Conference on Emerging Networking Experiments And Technologies*, pp. 144–150, 2019.

[16] "Let's Encrypt." 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] 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.

[19] N. Cardwell, Y. Cheng, C. S. Gunn, and all, "BBR: Congestion-Based Congestion Control," *Queue*, vol. 14, no. 5, p. 50, 2016.

[20] Y. Cheng *et al.*, "RFC 7413-TCP Fast Open," 2014.