How Good Is QUIC Actually?

Manuel Burghard, Benedikt Jaeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: burghard@in.tum.de, jaeger@net.in.tum.de

Abstract—**The Internet Engineering Task Force (IETF) is currently finalizing the standardization of the QUIC core transport protocol and HTTP/3 (H3) with a target date of July 2019 for the final specification. The working group has defined five key goals that should be improved on: connection establishment and transport latency, head-of-line blocking, secure transport, allowing future evolution, as well as multipath and forward error correction extensions.**

In this paper, we explain the problems QUIC tries to solve, discuss QUIC's proposed solutions, and highlight their strengths and weaknesses.

Index Terms—**networks, multi-layer transport protocol, latency reduction, performance**

1. Introduction

QUIC was initially designed by Roskind [1] at Google to improve the performance of the SPDY protocol running on top of the Transmission Control Protocol (TCP) and Transport Layer Security (TLS). SPDY was the conceptional predecessor of HTTP/2 (H2) which was standardized in 2015 [2]. QUIC merged key functionalities of the transport, security, and application layer into a new protocol based on the User Datagram Protocol (UDP) with the goal of reducing latency and improving performance [1]. In 2015, a first draft for standardization by the Internet Engineering Task Force (IETF) was submitted by Iyengar and Swett [3] and in 2016 a working group was formed [4]. To distinguish between the IETF's and Google's version of QUIC we refer to Google QUIC as gQUIC.

Langely et al. [5] state that Google used its control of the Chrome browser and its web services to create a large scale test and evaluation environment for the development of gQUIC. Initially, a small number of users were randomly selected to test gQUIC in Chrome and the amount was gradually increased until 2017 when only a small control group using TCP and TLS was left. The control over client and server software in combination with a large user base allowed fast and regression free iterations of the protocol [5].

IETF QUIC is a stream-multiplexing UDP-based protocol which always encrypts its payload using TLS 1.3. In contrast to Google's gQUIC implementation the IETF working group decided to separate gQUIC's transport protocol and the adopted H2 application protocol to allow other application level protocols [6]. The adapted version of H2 for QUIC is currently being standardized, too, and will be named HTTP/3 (H3) [7].

The QUIC working group defined five key goals QUIC should deliver [6]:

- Secure the transported payload using TLS 1.3.
- Enable deployment without requiring changes to network equipment along the path.
- Multiplexing without head-of-line blocking, like introduced by H2.
- Minimize the connection establishment and transport latency.
- Enable extensions for forward error correction and multipath connections.

This paper focuses on the IETF version of QUIC in its discussions, but references observations of real world data usage by gQUIC because it is the only large scale deployment as of June 2019.

The remaining parts of this paper are structured in the following way. In Section 2 an overview of related work is provided. Section 3 focuses on strengths and weaknesses of QUIC's key goals. For each goal, we will discuss the problems that should be solved, the solutions proposed by QUIC, and their strengths and weaknesses. Section 4 discusses the overall performance of QUIC in comparison to TCP. Section 5 concludes this paper and discusses future work.

2. Related Work

A short overview of QUIC is presented by Yosofie [8] and a more elaborate introduction of QUIC and H3 is given by Stenberg [9]. The full specification is currently available in a series of IETF drafts [7], [10]–[12]. A comparison between QUIC and Stream Control Transmission Protocol (SCTP) is presented by Joseph et al. [13].

Cook et al. evaluated the performance of QUIC in comparison to H2 based on the page load time [14]. Kakhki et al. [15] present the results of their performance comparison of TCP and QUIC using different environments and by running the tests with different versions of the QUIC implementation of the Chromium project to compare the development over time. QUIC was originally intended to be implemented in user space, but Wang et al. implemented QUIC in the Linux kernel and compared its performance to TCP [16].

In 2013 Multipath TCP (MPTCP) extensions were standardized by Ford et al. [17] which define the usage of multiple (disjoint) paths through a network to provide TCP's bi-directional stream. Cheng et al. defined TCP Fast Open (TFO) [18] which may reduce the connection establishment by one round-trip time (RTT) for TCP, but also introduces a number of new security implications, like resource exhaustion or amplified reflection attacks.

3. Strengths and Weaknesses

In this Section, we evaluate the strengths and weaknesses of the key goals set out by the QUIC working group. Each subsection focuses on one of the goals, explaining the existing problem and QUIC's proposed solution.

3.1. Secure Transport

The leaks and uncoverings by journalists of the last decade have proven those correct who always warned about the interest of governments and private corporations in private user data [19], [20]. Nowadays, it is broadly acknowledged that user data should be encrypted when it is sent over the Internet to ensure a minimum level of integrity, confidentiality and privacy. The HTTP protocol up to version two relies on TLS on top of TCP for its (optional) encryption. When opening a connection to a server, first the three-way handshake of TCP [21] is performed followed by the TLS handshake [22] to establish a secure bi-directional connection between a client and a server. During the standardization of H2, the working group discussed making always-on encryption part of the protocol, but was unable to find consensus on this topic [2], [23].

Google's first specification of QUIC from 2012 [1] already required always-on encryption for the payload and some header fields. More information about why some header fields are encrypted will be provided in Section 3.2. Google's initial design for QUIC predated TLS 1.3 and therefore Langley et al. came up with their own encryption [24]. The QUIC working group later decided to use TLS 1.3 for encryption [10], [12].

Always encrypting the payload benefits any potential end user of the protocol by keeping their communication private. It also forces companies or other organizations who wish to use QUIC to provide secure transport, which should lead to a broader adoption of encrypted communication on the Internet. On the other hand, there are companies like banks which have to meet certain regulatory or compliance requirements which effectively prevent the usage of TLS 1.3 due to being too secure and constrains these companies to standards like Enterprise Transport Security (ETS) [25], which was recently assigned a CVE number for its lack of per-session forward secrecy [26].

The usage of TLS 1.3 comes at a cost, too: QUIC is vulnerable to application layer replay attacks when using 0-RTT, similar to TLS 1.3, and application layer protocols must include mitigations [12].

3.2. Enabling Future Changes to QUIC

Iterating on a protocol like TCP or trying to introduce a new protocol like SCTP often requires long adoption time because the network equipment along the paths must support and understand the traffic. Especially router and firewall vendors are known to make certain assumptions about the protocols they are supporting, like dropping TCP packets which contain unknown or new TCP options [9]. Any unexpected change may lead to packets being flagged as illegitimate which results in rejecting or dropping said packets. For example, adoption of SCTP on top of IP is still not widespread today [13]. This stiffness of the existing Internet infrastructure is referred to as *ossification* [5], [9].

In addition to that, protocols like TCP or SCTP are usually implemented in the kernel and require the commitment of operating system vendors, changes to that implementation are bound to operating system updates, and users must install those updates until an application can rely on a new protocol. As of June 2019 SCTP is still not supported by Microsoft Windows or Apple's operating systems, but third party implementations exist [27].

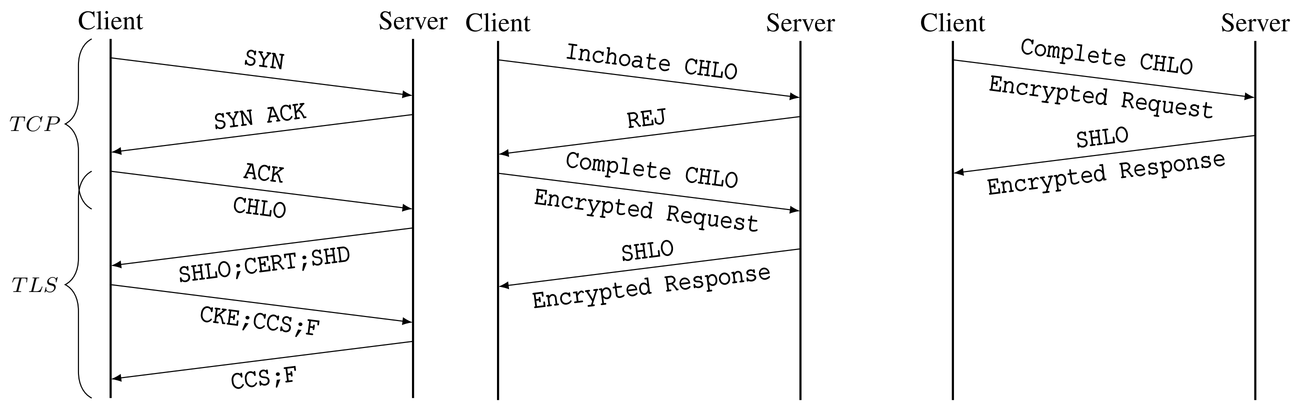
QUIC tries to solve the problem of adoption and allowing for future changes: First of all, it is built on top of UDP. Existing network equipment already supports UDP and does not require support for a completely new transport layer protocol. Second, QUIC can be fully implemented in the user space which makes it independent of any support in an operating system kernel. A downside of this approach is a potential performance penalty induced by system calls and context switches, but a user space implementation decouples development from the operating system release schedule and allows faster iteration and easier deployment to legacy systems. Applications can include a QUIC library and can distribute newer versions independent of the operating system they are running on, like Google is doing with Google Chrome. The current adoption rate also speaks for QUIC's approach: Langley et al. [5] estimated the amount of QUIC traffic on the Internet at 7% in 2017, about five years after the introduction. In addition to that, QUIC also includes version negotiation in the protocol as part of opening a connection [10], thereby enabling the introduction of new QUIC versions while allowing clients and servers to agree on a version supported by both sides.

To prevent ossification, QUIC tries to encrypt as much data as possible, including signaling information [10], to hide it from network equipment and prevent vendors of said equipment from making assumptions that will interfere or prevent future changes to the protocol.

The strengths of the solutions provided are obvious: The adoption rate is already significant and precautions have been taken to simplify changes to the protocol in the future. Nevertheless, 4.7% of video playback traffic is having problems due to network equipment blocking, rate limiting, or otherwise limiting UDP traffic [5].

3.3. Head-of-Line Blocking

Head-of-line (HOL) blocking describes the situation of sequential packets or requests being held up by the first item in a serial queue. HOL blocking can appear on different network layers like TCP when a packet is dropped/delayed or in HTTP/1.1 (H1.1) when all open TCP connections to a server are already transferring requests and additional request are forced to wait until another request has finished. With the introduction of H2 and its multiplexed streams the number of connections opened by e.g. a browser could be reduced to one compared to the up to six from H1.1 while also allowing multiple requests



(a) TCP and TLS handshake (simplified).

(b) QUIC 1-RTT connection establishment [5]

(c) QUIC 0-RTT connection establishment [5]

Figure 1: Comparison of TCP and TLS, QUIC 1-RTT, and QUIC 0-RTT handshakes.
Abbreviations used in the diagrams: ClientHello (CHLO), ServerHello (SHLO), Certificate (CERT), ServerHelloDone (SHD), ClientKeyExchange (CKE), ChangeCipherSpec (CCS), Finished (F), Reject (REJ)

being performed at the same time. Therefore, H2 may be less likely to suffer from application layer HOL blocking in comparison to H1.1, but is more prone to HOL blocking on the TCP layer [9], [28].

Similar to H2, QUIC supports multiple streams over a single connection, but loss detection and recovery are part of QUIC itself and not of an underlying protocol layer like TCP. UDP is connectionless and does not provide any loss detection or recovery at all. In case of lost packets, recovery only impacts the streams whose frames were part of the lost packets. Other streams are not affected by the recovery and hence not blocked. Retransmission is also different when compared to TCP. Instead of retransmitting a lost packet, QUIC checks for every lost stream frame whether the contained data is still needed. If that is the case, the stream frames and packets are then retransmitted. If a stream is reset in the meantime, the lost frames for this stream are not retransmitted [10], [11].

Eliminating HOL blocking leads to better performance of QUIC in lossy environments. The tests performed by Kakhki et al. [15] support this claim for desktop and mobile environments, although the gains are inferior in the mobile environment due to slower packet consumption of QUIC's userspace implementation. The downside is that loss detection and recovery were reimplemented on top of UDP although TCP implementations already exist and are well tested.

3.4. Connection Establishment and Transport Latency

The main goal set out initially by Google was to reduce latency, especially for connection establishment which usually includes some form of handshake, like the well known TCP three-way handshake [1]. For an H2 connection to a server, first a TCP connection is opened, followed by a TLS handshake if an encrypted connection is desired. The full handshake flow for TCP and TLS is shown in Figure 1a and results in a minimum of three round trips until a connection is established and application data can be transferred. In an high delay environment, the handshake latency can highly influence the perceived performance of the network connection and thereby degrade the user experience.

QUIC improves on connection establishment by combining the transport and cryptographic handshake [10]. This results in a 1-RTT handshake which means only one round-trip is needed until application data can be transfered using the newly established connection. Furthermore, QUIC also supports 0-RTT handshakes, which use cryptographic information from a previous connection for even faster connection establishment allowing data transfer to start with the initial message from a client. Both handshake variants can be seen in Figure 1b and 1c. Data released by Google on handshake latencies shows that TCP and QUIC handshake latencies are growing linearly with growing RTTs but QUIC's slope is lower due to 0-RTT [5]. Even when considering just the 1-RTT connection establishments, QUIC only takes about half the time of TCP. The benefit of supporting 0-RTT handshakes was further shown by [15] who compared QUIC with and without enabled 0-RTT handshakes and found significantly improved performance for small transfers up to 10 kB. There was an attempt to reduce the handshake latency for TCP with TFO in [18], but according to Paasch it suffers from ossification and the success rate is at about 80% [29]. Google's servers support TFO but the influence of TFO for their handshake comparison is not shown or highlighted [5].

TLS 1.3 support for 1-RTT and 0-RTT handshakes is not limited to QUIC and can can be used in the underlying connection of an H2 stack to further improve the connection establishment latency in the classic HTTP stack. This could diminish the performance gains of QUIC in comparison to TCP and TLS.

3.5. Multipath and Forward Error Correction

QUIC's multipath extension is not part of the July 2019 milestone but adoption is scheduled for December 2019 and the handover to the Internet Engineering Steering Group (IESG) is scheduled for May 2020 [6]. De Coninck et al. present an experimental implementation of QUIC multipath with the goals of resilience to connection

	Search latency	Video latency	Video rebuffer rate
Desktop	8.0	8.0	18.0
Mobile	3.6	5.3	15.3

TABLE 1: Mean percent reduction of the search latency, video latency, and video rebuffer rate observed by [5].

failures and the combination of available resources, and compare its performance to MPTCP [30]. They are also the authors of the current IETF draft for QUIC multipath [31]. Forward Error Correction (FEC) is out of scope of the first standardization [6]. Google supported FEC in gQUIC but removed it in 2016 due to the unconvincing results and the increase in code complexity [5]. Due to these reasons we do not further discuss multipath and FEC.

4. Overall Performance

In this section we want to compare different reports about QUIC's performance. We focus on three papers starting with Langley et al. [5].

In 2017 Google presented the results and measurements of their production deployment of QUIC for all Google services with probably the largest sample size [5]. Their observations of handshake latencies were already summarized in Subsection 3.4. Google evaluated the performance of QUIC in comparison to TCP and TLS using the search latency, video latency, and video rebuffer rate as metrics. We only summarize the mean percent reduction in Table 1 and refer to [5] for more details. Clearly visible in the data provided are the inferior gains of QUIC in the mobile environment. This is explained by a lower 0-RTT handshake rate due to mobile devices changing IP addresses when switching networks and, thereby invalidating the cached cryptographic information for the handshake. Another reason for a lower 0-RTT handshake rate is hitting different servers causing a 0-RTT handshake to fail, too. The results for video latency are additionally influenced by the YouTube app which performs handshakes in the background to improve latency. The video rebuffer rate measures the time spent on rebuffering data during video playback and is not directly dependent on the handshake latency. Instead, it depends on loss recovery and overall throughput of the established connection. According to Langley et al. QUIC performs best in high delay, low bandwidth, and lossy networks. This claim is further supported by comparing QUIC's performance when used in India to the performance in South Korea with both countries being on opposite ends of Internet quality scale with regard to delay, loss, and throughput. During the tests, they also noticed higher CPU usage for QUIC. Even after optimizations QUIC doubled TCP and TLS CPU usage. Stenberg [9] mentions the slowness and higher CPU consumption of QUIC, too, and explains it with the lack of hardware acceleration and lack of optimized UDP stacks.

Cook et al. [14] evaluated QUIC's performance based on the Page Load Time (PLT) of websites in different scenarios with regard to delay, loss, network type (cellular vs ADSL), and network load in comparison to H1.1 and H2 on top of TCP and TLS. The test environment consisted of copies of websites hosted on virtual machines and their real world counterparts. Their tests showed that QUIC performs better under delay, in lossy networks, and when connected to a cellular network. These results match with the observations of Langley et al. described above. In addition to the PLT evaluation, the influence of the distribution of a website was investigated showing that QUIC performs better if the number of servers hosting website resources is low.

Another evaluation of QUIC's performance was done by Kakhki et al. [15] using Chrome and the corresponding server component [32]. The scenarios compared QUIC to TCP based on PLT in a desktop and mobile environments, video streaming performance, the fairness of QUIC with regard to sharing of bottleneck bandwidth, and the impact of in-network proxying when using QUIC. The performance characteristics of QUIC for the PLT match with the results we presented before: QUIC performs better than TCP for small objects and connections with loss, the gains on the desktop are higher than those on mobile devices. Interesting findings include QUIC's poor performance when packet reordering is required, resulting in TCP outperforming QUIC. This is explained by the threshold for negative-acknowledgments being based on a fixed number in QUIC causing it to start retransmission whereas TCP benefits from dynamically adapting the threshold. During the setup of the testbed, the authors noticed that the QUIC server's default parameters result in worse performance when compared to Google's production QUIC servers and therefore tweaked the parameters until the performance matched that of Google's servers. An unfairness of QUIC was observed when comparing the consumption of bottleneck bandwidth: QUIC vs QUIC results in equal shares, but QUIC vs TCP results in an unfair imbalance towards QUIC although both used the same congestion control algorithm.

5. Conclusion and Future Work

One interesting area of future work is to compare QUIC's multipath capabilities to those of MPTCP when the associated milestone is done by the IETF working group. MPTCP is already implemented in major operating systems like Linux [33]. Additional future work may focus on performance comparisons of H2 over TCP and TLS, especially how 1-RTT and 0-RTT handshakes of TLS 1.3 influence the results.

The design of QUIC is the logical consequence of combining the benefits of recent network protocols and continuing good ideas a step forward. Always-on encryption is not just for the benefit of the end user's privacy, but prevents network ossification. The possibility of an user space implementation allows deployment independent of any minimal operating system version and rapid iterations. Faster handshakes and the elimination of HOL blocking lead an improved user experience. Overall, the strengths mentioned in this paper outweigh the weaknesses. QUIC may not replace TCP and TLS immediately, but there are areas like high latency and lossy networks or cellular networks where QUIC is well suited to take over.

References

[1] J. Roskind, "Quic: Multiplexed stream transport over udp," https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34, 2012, [Online: accessed 10-June-2019].

[2] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540, May 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7540.txt

[3] J. Iyengar and I. Swett, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2," https://tools.ietf.org/id/draft-tsvwg-quic-protocol-00.txt, 2017, [Online; accessed 04-June-2019].

[4] M. Westerlund and S. Dawkins, https://datatracker.ietf.org/doc/charter-ietf-quic/00-00/, 2016, [Online; accessed 04-June-2019].

[5] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 183–196. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098842

[6] Internet Engineering Task Force, "QUIC (quic)," https://datatracker.ietf.org/wg/quic/about/, [Online; accessed 04-June-2019].

[7] M. Bishop, "Hypertext Transfer Protocol Version 3 (HTTP/3)," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-20, Apr. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-20

[8] M. Yosofie, "Recent progress on the QUIC protocol," in *Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM), Winter Semester 2018/2019*, ser. Network Architectures and Services (NET), G. Carle, S. Günther, and B. Jaeger, Eds., vol. NET-2019-06-1. Munich, Germany: Chair of Network Architectures and Services, Department of Computer Science, Technical University of Munich, Jun. 2019, pp. 77–81.

[9] D. Stenberg, "HTTP/3 explained," https://http3-explained.haxx.se/en/, 2018, [Online; accessed 04-June-2019].

[10] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-transport-20, Apr. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-20

[11] J. Iyengar and I. Swett, "QUIC Loss Detection and Congestion Control," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-recovery-20, Apr. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-recovery-20

[12] M. Thomson and S. Turner, "Using TLS to Secure QUIC," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-tls-20, Apr. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-tls-20

[13] A. Joseph, T. Li, Z. He, Y. Cui, and L. Zhang, "A Comparison between SCTP and QUIC," Internet Engineering Task Force, Internet-Draft draft-joseph-quic-comparison-quic-sctp-00, Mar. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-joseph-quic-comparison-quic-sctp-00

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

[15] 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," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17. New York, NY, USA: ACM, 2017, pp. 290–303. [Online]. Available: http://doi.acm.org/10.1145/3131365.3131368

[16] P. Wang, C. Bianco, J. Riihijärvi, and M. Petrova, "Implementation and performance evaluation of the quic protocol in linux kernel," in *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWIM '18. New York, NY, USA: ACM, 2018, pp. 227–234. [Online]. Available: http://doi.acm.org/10.1145/3242102.3242106

[17] A. Ford, C. Raiciu, M. J. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824, Jan. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc6824.txt

[18] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain, "TCP Fast Open," RFC 7413, Dec. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7413.txt

[19] G. Greenwald, "NSA collecting phone records of millions of Verizon customers daily," Jun. 2013, [Online; accessed 21-June-2019].

[20] K. Roose, "Facebook Emails Show Its Real Mission: Making Money and Crushing Competition," Dec. 2018, [Online; accessed 21-June-2019].

[21] "Transmission Control Protocol," RFC 793, Sep. 1981. [Online]. Available: https://rfc-editor.org/rfc/rfc793.txt

[22] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8446.txt

[23] "Http/2 frequently asked questions," https://http2.github.io/faq/#does-http2-require-encryption, [Online: accessed 10-June-2019].

[24] A. Langley and W.-T. Chang, "Quic crypto," https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g, 2013, [Online: accessed 10-June-2019].

[25] T. Rutkowski and S. Compans, "ETSI TS 103 523-3," https://www.etsi.org/deliver/etsi_ts/103500_103599/10352303/01.01.01_60/ts_10352303v010101p.pdf, European Telecommunications Standards Institute, Technical Specification, Oct. 2018, [Online; accessed 16-June-2019].

[26] "CVE-2019-9191," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9191, 2019, [Online; accessed 23-June-2019].

[27] M. Tüxen, "usrsctp," https://github.com/sctplab/usrsctp, [Online: accessed 15-June-2019].

[28] D. Stenberg, "http2 explained," https://daniel.haxx.se/http2/, 2014, [Online; accessed 15-June-2019].

[29] C. Paasch, "Network Support for TCP Fast Open," 2016, presented at NANOG67 in Chicago, IL. [Online]. Available: https://archive.nanog.org/sites/default/files/Paasch_Network_Support.pdf

[30] Q. De Coninck and O. Bonaventure, "Multipath quic: Design and evaluation," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '17. New York, NY, USA: ACM, 2017, pp. 160–166. [Online]. Available: http://doi.acm.org/10.1145/3143361.3143370

[31] Q. D. Coninck and O. Bonaventure, "Multipath Extensions for QUIC (MP-QUIC)," Internet Engineering Task Force, Internet-Draft draft-deconinck-quic-multipath-03, Aug. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-deconinck-quic-multipath-03

[32] "The chromium projects," https://www.chromium.org, [Online; accessed 18-June-2019].

[33] "MultiPath TCP - Linux Kernel implementation," http://multipath-tcp.org/pmwiki.php/Main/HomePage, [Online; accessed 16-June-2019].