Going Beyond TCP: QUIC, The Novel Transport Protocol*

André Rosa
DI/FCT/NOVA University of Lisbon, Lisboa, Portugal
Student № 48043
af.rosa@campus.fct.unl

***Abstract*—The ever-increasing amount of latency-sensitive web services has lead to a demand for minimizing data secure transport's latency, over the Internet. Nowadays, the majority of this transport is done by leveraging HTTPS. However, this protocol presents some performance limitations, being its implementation over TCP its main bottleneck. Unfortunately, due to IP middleboxes becoming dependent on the conventional transport protocols, it limited the ability to modify TCP and to deploy new transport protocols. As such, to address the need in reducing the latency of data transport and to circumvent the evolutionary limitations of TCP, a new application-level transport protocol, called QUIC, emerged.**

QUIC enables achieving a 0-RTT handshake and provides an abstraction of parallel streams over the same connection while avoiding HOL blocking. In addition to this, QUIC addresses other performance limitations of TCP. As such, this new protocol has been gaining a lot of relevance, currently being standardized and included in the new HTTP version 3. Therefore, in this paper, we review the QUIC protocol and present a summary of some experimental evaluations on its performance, found in the literature. Finally, we discuss whether QUIC will replace TCP as the main transport protocol on the Internet, in the near future.

***Index Terms*—QUIC, TCP, UDP, HTTP/3, HTTPS**

I. Introduction

The surge of latency-sensitive web services is leading to an ever-increasing demand for minimizing data transport's latency on the Internet. Besides, it is becoming more frequent that this transport is done securely, which can incur new sources of delay. Nowadays, the vast majority of secure data transport is done by leveraging the Hyper Text Transfer Protocol Secure (HTTPS). This protocol consists of layering the Hyper Text Transfer Protocol (HTTP) over the Transport Layer Security (TLS) protocol, which, in turn, is layered over the well-known Transmission Control Protocol (TCP).

In the past, there were already endeavors in seeking to improve the HTTPS' stack performance. SPDY[1] [1], a protocol developed by Google which was later standardized as HTTP/2 [1], emerged as a faster alternative to the conventional HTTP/1.1 [2]. Although with some limitations, new amendments have also been proposed to TCP and TLS to reduce their handshake latency: TLS 1.3 [3] with TCP Fast Open (TCP-FO) [4], [5]. However, TCP remains as the main bottleneck in the data transmission's performance.

This document was written in the context of the APRC course in replacement of the first midterm test.

[1]Pronounced as "speedy"

Unfortunately, there are several limitations in modifying TCP and, although new transport protocols have already been proposed to address the diversity of application requirements better than TCP is capable, such as the Stream Control Transmission Protocol (SCTP) [6] and the Datagram Congestion Control Protocol (DCCP) [7], their deployment is also limited [8]. Both of these challenges arise due to several Internet Protocol's packet forwarding middleboxes[2] have become intrinsically dependent on the conventional transport-level protocols: TCP and UDP (User Datagram Protocol). The cause of this dependence is that most of the middleboxes inspect or modify the transport protocol's headers. For instance, firewalls filter everything unfamiliar, such as new transport protocols; and Network Address Translation (NAT) modifies transport protocol's ports. Consequently, this not only removes flexibility in the modification of these protocols, a process known as *ossification*, but also difficults the introduction of new protocols at this level, since it is necessary for middleboxes adding explicit support for them.

Thus, in order to address the need to reduce the latency of transport and to circumvent the evolutionary limitations of current protocols, new application-level transport protocols started to emerge [9]–[13]. Among them, one particular novel protocol has been gaining a lot of relevance: QUIC[3] [14], [15]. This protocol, developed by Google, is a secure connection-oriented protocol that aims to be a faster and more flexible alternative to the conventional HTTPS stack.

Therefore, in this paper, we inside on the QUIC protocol, discussing the decisions behind its design as well as highlighting the limitations of the current HTTPS stack that motivated those decisions. The remainder of this paper is structured as follows: Section II presents an overview of QUIC, highlighting its mechanisms that address the limitations of the current HTTPS stack; Section III presents a literature review of some experimental evaluations to measure QUIC's performance as well as the results of studies on the wide-spread deployment of QUIC; Section IV delves the ongoing standardization of QUIC and its role on HTTP/3; Section V discusses the future of TCP in face of the emergence of QUIC; and finally, Section VI finishes the paper with some final remarks.

[2]Middleboxes are any intermediary in IP packets forwarding, such as routers and firewalls.

[3]In the standardized version of QUIC, the term "QUIC" is a name, not an acronym. Originally, QUIC stood for Quick UDP Internet Connections.

II. QUIC Protocol

In this section, we discuss the architecture of QUIC, highlighting its components while relating them to the challenges and limitations of the HTTPS stack that they address.

A. Packet Structure

As previously mentioned, middleboxes lead to TCP ossification, limiting its flexibility in adapting to new requirements and challenges, and are the reason for the difficulty in deploying new transport protocols. Furthermore, TCP is implemented as part of the Operating System (OS), which constraints quickly deploying new versions.

To circumvent these limitations, QUIC was designed to operate at the application level, over UDP. The benefits of this decision are two-fold. First, since the origin of UDP dates back to the early days of the Internet, it is guaranteed to be compatible with conventional middleboxes. Secondly, the fact that it operates at the application level eases the deployment of new versions, independent of the underlying OS. To mitigate ossification even further, QUIC also encrypts and authenticates portions of its header and not just the payload, which mitigates middleboxes consulting and tampering the packets. However, there are some parts of the header which are not encrypted since they are needed for identification, routing, or decryption purposes. One of such parts is the *Connection ID*, a 64-bit number which uniquely identifies each connection in contrast to the 5-tuple of TCP: destination's and source's IP address/port pairs, and TCP's protocol number. This allows connections to survive IP addresses and ports changes. Such changes may occur, for instance, due to NAT timeouts and rebindings, much more frequent in UDP than in TCP, or the clients changing networks.

B. Connection Establishment

The TCP handshake incurs in at least one round-trip time (RTT) to configure and establish a connection, before any data from upper layers can be transmitted. Layering TLS over TCP increases the overall HTTPS connection handshake delay with two more RTTs.

Contrastingly, QUIC combines both the transport and cryptographic handshakes, requiring only one RTT before any data can be transmitted. This handshake is decomposed into two parts: the initial handshake and the repeat handshake.

The initial handshake exchanges long-term information between the client and the server, such as certificate chains and other server configurations, and establishes a preliminary secure transport connection, requiring only 1 RTT. However, if the client is using a version not supported by the server, a version negotiation happens which increases the initial handshake delay to 2 RTTs. Due to the rapid development of new QUIC versions, this may become rather common. Upon the end of the initial handshake, the client caches the server's information and the *source address token*, a cryptographic token, authenticated by the server, containing the client's public IP address and a timestamp. This token serves to prove to the server that the client is the owner of the IP address which performed the initial handshake, and is appended in all future requests. Furthermore, it can also later be used to quickly establish a new connection to the same *origin* [16]. In short, an origin is an identifier of a protection domain and usually corresponds to a triple containing a scheme, host, and port.

Leveraging the initial connection, the repeat handshake establishes the final secure connection and requires 0 setup RTTs, since data can be transported at the same time. On future connections to the same origin, the client can attempt to quickly establish a new 0-RTT connection, by performing a repeat handshake containing the *source address token*, among other information. In the case this attempt fails, a new initial handshake is performed.

Inspired by this handshake, TLS 1.3 [3] also introduced 1-RTT and 0-RTT handshakes, leveraging TCP Fast Open (TCP-FO) [4], [5], which allows piggybacking data in the SYN message. However, unlike QUIC, the amount of data that can be piggybacked is limited.

C. Stream Multiplexing

In HTTP/1.1, several TCP connections can be used simultaneously to perform parallel transferences of multiple requests and replies. However, the number of such connections should be limited to avoid incurring high latency and overhead. Whenever this maximum number of allowed connections is reached, further transferences are postponed until some on-going transference is completed, which liberates a TCP connection. This can be a major source of delay for HTTPS and is an instance of a phenomenon called *head-of-line Blocking (HOL blocking)*. To tackle this limitation, HTTP/2 [1] establishes a single TCP connection, between clients and servers, which is multiplexed by several simultaneous transferences. As such, several transferences can simultaneously be performed, while avoiding HOL blocking. Nevertheless, HOL blocking can still occur within TCP itself. Whenever a TCP segment is lost, the following segments are not transmitted while the current segment is not successfully delivered. If the lost segment contains data of a given transference, all the other multiplexed transferences become blocked unnecessarily. Furthermore, when segments are received out of order, they are buffered and are only delivered after all the previous segments. This behavior can prevent some transferences from being completed, although all their data might already have been received by the destination.

To address these challenges, QUIC provides a set of reliable bidirectional byte stream abstractions called *streams*, that are divided into a sequence of *frames* (akin to TCP segments), which are then multiplexed in a single connection. Each datagram transmitted can contain several frames of different streams. However, in QUIC, the loss and arrivals out of order of datagrams, only affect streams containing data on those datagrams, mitigating HOL blocking. Thus, QUIC provides an abstraction similar to simultaneous TCP connections to the same destination, yet more efficient than leveraging a single TCP connection to provide the same abstraction.

D. Loss Detection

In TCP, whenever a segment is lost, it is retransmitted with the same sequence number of the original transmission. Consequently, the receiver of an acknowledgment (ACK) for that sequence number is not able to determine if the ACK was due to the original transmission, assumed as being lost, or subsequent retransmission. This phenomenon is known as the TCP *retransmission ambiguity problem*. As such, assuming that the ACK was for the latest retransmission may skew RTT estimations, which are used to determine the retransmission timeouts. Although there were already proposed solutions to mitigate skewed RTT estimations, such as Karn's Algorithm [17], they are far from being perfect.

To not incur the same problem, in QUIC, each transmitted packet has a new sequence number, independently if it is a retransmission of an older packet or not. This enables distinguishing the packet that caused each ACK, leading to more precise RTT estimation and thus more accurate loss detection than in TCP. Nevertheless, this sequence number can no longer be used to order deliveries. For this, each stream frame has its own sequence number.

Additionally, QUIC allows up to 256 simultaneous ACKs, having more capacity than TCP with Selective ACKs (SACKs) [18], and thus being more robust to packet losses and reorderings.

E. Flow Control

In transmission protocols, to avoid senders transmitting data faster than it is consumed by receivers, there are applied flow control mechanisms. In TCP, flow control mechanisms only have to address a single flow of data. However, in QUIC, due to the usage of stream multiplexing, a stream consumed more slowly than the others, could potentially occupy the majority of the receive buffer. Consequently, this would prevent the sender from transmitting data from other streams, leading to HOL blocking. To address this, QUIC employs flow control at two different levels: *connection-level flow control* and *stream-level flow control*. On the one hand, *connection-level flow control* restricts the overall amount of data that the sender can transmit, across all streams. On the other hand, *stream-level flow control* restricts the amount of data that the sender can transmit, for each stream. Although being applied at different abstractions, both mechanisms, however, work in the same way, employing a credit-based flow control. In it, each receiver periodically announces the number of bytes it is capable of receiving, either on each stream or the overall connection. According to this information, the sender adjusts the amount of data it transmits.

F. Congestion Control

In transmission protocols, to avoid congesting the network, congestion control mechanisms are applied. TCP has several congestion control algorithms, proposed over the years [19]. Similarly, QUIC does not specify a particular congestion control algorithm and allows to easily integrate any external algorithm, such as any TCP Congestion Control algorithm. However, during QUIC's development, it utilized the Cubic congestion control algorithm [14].

G. QUIC Discovery

Whenever a client desires to communicate with a server through HTTPS over QUIC, it has first to find out whether the server accepts QUIC connections. For this, [14] authors propose that clients leverage TLS/TCP to issue the first request to a given origin for the first time. In case the server accepts QUIC connections, it appends a header "alt-svc" in the HTTP reply, which indicates to the client its availability in receiving QUIC connections. Next, to guarantee that QUIC (or UDP) is not blocked on the path between the server and the client, the later initiates both a QUIC and TLS/TCP connections sequentially. The protocol which successfully establishes a connection first is the one that will further be leveraged in subsequent requests. This, however, is not an ideal solution for two reasons. First, it incurs in the conventional HTTPS limitations on the first request to a given origin which supports QUIC. Second, in case QUIC is not blocked along the path, it wastes server's resources by attempting to establish two simultaneous connections, one for QUIC and the other for TCP.

III. Experimental Evaluation

In this section, we summarize several experimental evaluations on QUIC, found in the literature. We start by discussing performance evaluations and continue with wide-spread deployment studies of QUIC.

A. Performance Evaluation

In [14] Google performed a preliminary experimental evaluation of QUIC on a real Internet-scale deployment, with real clients. For this, they added support for QUIC to their desktop web browser Chrome and their mobile applications Google Search and YouTube; as well as several of their servers. The authors of the experiments focused on measuring service-specific metrics instead of conventional transport metrics, such as maximum throughput or loss recovery. This decision was due that service-specific metrics measure the usefulness of the protocol for applications and not just if the protocol is working as intended. Three metrics were chosen, which stand for diverse transport use cases: search latency for a low-load latency-sensitive application; and video latency and rebuffer rate for a heavy-load bandwidth-sensitive application. The average of the results is summarized next:

- Search latency corresponds to the elapsed time between a user entering a search term and the delivery of the corresponding result. QUIC achieved an improvement over HTTP/2 (no TCP-FO) of 8.0% for browser connections and 3.6% for the search mobile application. These improvements were mainly due to QUIC's 0-RTT handshake, corresponding to 88% (68%) of all browser (search mobile application) handshakes.
- Video latency corresponds to the time between the user clicking to play the video and the video start playing.

QUIC performed better than HTTP/1.1 with 2 simultaneous TCP (no TCP-FO) connections, with an improvement of 8.0% for browser connections and 5.3% for the search mobile application. These improvements were also mainly due to QUIC's 0-RTT handshake, corresponding to 85% (65%) of all browser (YouTube mobile application) handshakes.

- Video Rebuffer Rate is the percentage of time that a video remains paused to rebuffer data. QUIC performed better than HTTP/1.1 with 2 simultaneous TCP (no TCP-FO) connections, with an improvement of 18.0% for browser connections and 15.3% for the YouTube mobile application. These improvements were due to lower loss-recovery latency and higher overall throughput.

Overall, QUIC performed better than HTTP/1.1 and HTTP/2, being its benefits higher, the higher the congestion, packet loss, or RTT. However, the latency results can be miss-leading, and a more fair comparison would have been against HTTP 1.1 and 2 over TLS 1.3 with TCP-FO since it also enables achieving 0-RTT connections. In the same study, the authors pointed out that the CPU load of QUIC on servers was double when compared to TCP.

Opposing to the previous results, in [20], the authors observed that for Dynamic Adaptive Streaming over HTTP (DASH), TCP presented significantly better performance than QUIC. While using TCP, the average bit-rate was significantly higher than when resorting to QUIC. However, QUIC presented a smaller degree of quality changes. These results were attributed to most quality adaptation algorithms being optimized for TCP and not leveraging QUIC's full potential.

In contrary to both the two previous studies, the authors of [21] claim that "no significant difference of application-level QoE factors between QUIC and TCP could be observed" for video streaming and web browsing sessions. Furthermore, they also determined that the difference between the initial delays of TCP and QUIC presented an almost uniform distribution of mean 0, i.e. on average there was no difference in initial delays for the applications. Nonetheless, there were no mentions if TCP-FO was being used or not.

B. *Deployment Evaluation*

Since its early conception in 2012, QUIC has been quickly gaining adoption. In 2017, Google stated that over 30% of its out-going traffic was QUIC, which corresponded to an estimation of 7% of all internet traffic [14]. Two years later, in 2019, those estimations rose up to 42.1% and 9.1%, respectively [22].

However, an extensive analysis of the IPv4 address space revealed that only around 617 590 IP addresses supported QUIC and that almost the totality (94.24%) of those addresses belonged only to two companies: Google and Akamai [22]. A more detailed analysis on the availability of QUIC support was performed on 46% of the DNS namespace and revealed that 161 thousand domains supported QUIC, however, only 20.5% presented similar content to conventional HTTPS, only 9.3% of them presented valid certificates, and only 6.8% enabled QUIC Discovery over HTTPS (§II-G).

Therefore, although QUIC has been increasing its traffic share, its wide-spread adoption is still very limited. Perhaps, with the introduction of the new HTTP/3 standard, discussed in §IV, this may quickly change.

IV. Hypertext Transfer Protocol Version 3 (HTTP/3)

At present, the QUIC protocol is going through a standardization process by the Internet Engineering Task Force (IETF). One of the changes proposed to the QUIC version presented in this document is the integration with TLS 1.3 [23], to replace the cryptographic functionalities of QUIC.

Furthermore, a new version of HTTP (HTTP/3) [24] is also being standardized, consisting of a mapping of the HTTP semantics over QUIC with TLS 1.3. This new version of HTTP is heavily based on the previous, HTTP/2 [1], being some of its features subsumed by QUIC. In short, HTTP/3 also relies on binary framing of HTTP messages (requests and replies) before transmission and also leverages connection multiplexing to transmit several of these messages simultaneously. These mechanisms are discussed next.

A. *QPack*

Like HTTP/2, HTTP/3 resorts to header compression for efficiently representing HTTP header fields. For this, HTTP/3 uses *QPack* [25], its standard compression format. QPack uses two tables, one static and the other dynamic, which are addressed separately and associate header fields with their correspondent indices. The static table is preestablished and contains conventional header fields. On the other hand, the dynamic table is constructed during a connection. HTTP/3 leverages encoders and decoders to compress and decompress header lists, respectively. Encoders transform a header list into a header block, which preserves the order of each field. In turn, each field can be represented in two different ways: index to either the static or dynamic table; or literal representation. QUIC unidirectional streams can be used to send instructions between the encoder and the decoder on the other extreme of the connection, for instance, to insert a new entry on the dynamic table. However, since QUIC does not guarantee any delivery order of data belonging to different streams, a decoder might try to decode a field for which the correspondent entry is not yet on the dynamic table, leading to HOL blocking.

B. *Message Multiplexing*

Unlike HTTP/2, HTTP/3 does not implement connection multiplexing and instead delegates it to the underlying QUIC protocol. The HTTP/3 specification defines three stream types: request stream, push stream, and control stream. Request streams are used to transport regular HTTP messages, i.e. requests and responses. Each HTTP message comprises: i) the message's header, sent as a single HEADERS frame; ii) the payload's body, if the message has a payload, sent as a series of DATA frames; and iii) trailing headers, if pretended, sent as a single HEADERS frame. Transmissions on any other order correspond to an HTTP error. Upon a request, servers are allowed to reply with zero or more non-final responses followed by a final one, however, non-final responses cannot contain a body or trailing headers. Servers can also proactively send responses, called PUSH Responses, before clients issue a request, in a push stream (a server-initiated unidirectional QUIC stream). Upon establishing a connection, each side must create a single control stream and send its configuration parameters.

Each request must be sent on a new QUIC stream, and its consequent responses must be sent on the same stream. Immediately after sending a message on a stream of any type, the sender must close the sending part of the stream.

V. The Future of TCP

With the introduction of QUIC, which addresses TCP's limitations and provides greater flexibility in its evolution, the following questions arise: Is QUIC going to obsolete TCP and eventually take its place as the main bidirectional reliable transmission protocol on the Internet? Will TCP only be suitable for compatibility with legacy applications?

As of now, the results of experimental comparisons between QUIC and TCP are very contradicting and do not ultimately determine one protocol as being significantly better over the other. Regarding the protocol's design, although QUIC has greater flexibility to improve, only in very specific aspects it is clearly more robust than TCP, e.g. better packet loss detection. However, for the time being, the real impact of these specific aspects in its performance is not clear enough and further and more detailed experimental comparisons between the two protocols needs to be conducted.

On the other hand, the Internet evolves towards improving network conditions, such as bandwidth, decreasing losses, and even decreasing latency (bringing data and computations closer to end-users [26]). Thus, even if the QUIC emerges as a slightly better protocol, the performance differences may tend to decrease as the Internet infrastructure continues to evolve. Consequently, since the vast majority of distributed applications are built on top of TCP and since that, for now, there is no clear advantage of migrating to QUIC, there seems to not exist enough motivation to abandon TCP in the near future.

VI. Final Remarks

In this paper, we discussed QUIC, a novel application-level transport protocol developed to address the performance limitations of the conventional HTTPS stack. Its most appealing design features are the 0-RTT handshake and the multiplexing of several streams over a single connection, without incurring in HOL blocking.

Although QUIC was designed to be more robust than TCP, the preliminary experimental comparisons between the two protocols were very contradicting and did not reveal one protocol as being significantly better over the other. As such, further and more detailed experimental comparisons between the two protocols needs to be conducted.

Finally, we discussed whether QUIC will replace TCP as the main transport protocol on the Internet, and concluded that, for the time being, there seems to not exist enough motivation to abandon TCP, in the near future.

References

[1] 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

[2] R. T. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing," RFC 7230, Jun. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7230.txt

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

[4] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, "Tcp fast open," in *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/2079296.2079317

[5] 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

[6] R. Stewart and C. Metz, "Sctp: new transport protocol for tcp/ip," *IEEE Internet Computing*, vol. 5, no. 6, pp. 64–69, 2001.

[7] S. Floyd, M. J. Handley, and E. Kohler, "Datagram Congestion Control Protocol (DCCP)," RFC 4340, Mar. 2006. [Online]. Available: https://rfc-editor.org/rfc/rfc4340.txt

[8] K. Grinnemo, T. Jones, G. Fairhurst, D. Ros, A. Brunstrom, and P. Hurtig, "Towards a flexible internet transport layer architecture," in *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2016, pp. 1–7.

[9] B. Ford, "Structured streams: A new transport abstraction," in *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 361–372. [Online]. Available: https://doi.org/10.1145/1282380.1282421

[10] W. M. Petullo, X. Zhang, J. A. Solworth, D. J. Bernstein, and T. Lange, "Minimalt: Minimal-latency networking through better security," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 425–438. [Online]. Available: https://doi.org/10.1145/2508859.2516737

[11] Y. Gu and R. L. Grossman, "Udt: Udp-based data transfer for high-speed wide area networks," *Computer Networks*, vol. 51, no. 7, pp. 1777 – 1799, 2007, protocols for Fast, Long-Distance Networks. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128606003057

[12] M. Sharabayko, M. Sharabayko, J. Dube, J. Kim, and J. Kim, "The SRT Protocol," Internet Engineering Task Force, Internet-Draft draft-sharabayko-mops-srt-00, Mar. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-sharabayko-mops-srt-00

[13] M. R. Meiss *et al.*, "Tsunami: A high-speed rate-controlled protocol for file transfer," *Indiana University*, 2004.

[14] 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: Association for Computing Machinery, 2017, p. 183–196. [Online]. Available: https://doi.org/10.1145/3098822.3098842

[15] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-transport-27, Feb. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-27

[16] A. Barth, “The Web Origin Concept,” RFC 6454, Dec. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6454.txt

[17] P. Karn and C. Partridge, “Improving round-trip time estimates in reliable transport protocols,” *SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 1, p. 66–74, Jan. 1995. [Online]. Available: https://doi.org/10.1145/205447.205455

[18] S. Floyd, J. Mahdavi, M. Mathis, and D. A. Romanow, “TCP Selective Acknowledgment Options,” RFC 2018, Oct. 1996. [Online]. Available: https://rfc-editor.org/rfc/rfc2018.txt

[19] E. Blanton, D. V. Paxson, and M. Allman, “TCP Congestion Control,” RFC 5681, Sep. 2009. [Online]. Available: https://rfc-editor.org/rfc/rfc5681.txt

[20] D. Bhat, A. Rizk, and M. Zink, “Not so quic: A performance study of dash over quic,” in *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 13–18. [Online]. Available: https://doi.org/10.1145/3083165.3083175

[21] M. Seufert, R. Schatz, N. Wehner, B. Gardlo, and P. Casas, “Is quic becoming the new tcp? on the potential impact of a new protocol on networked multimedia qoe,” in *2019 Eleventh International Conference on Quality of Multimedia Experience (QoMEX)*, 2019, pp. 1–6.

[22] J. Rüth, I. Poese, C. Dietzel, and O. Hohlfeld, “A first look at quic in the wild,” in *Passive and Active Measurement*, R. Beverly, G. Smaragdakis, and A. Feldmann, Eds. Cham: Springer International Publishing, 2018, pp. 255–268.

[23] M. Thomson and S. Turner, “Using TLS to Secure QUIC,” Internet Engineering Task Force, Internet-Draft draft-ietf-quic-tls-27, Feb. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-tls-27

[24] M. Bishop, “Hypertext Transfer Protocol Version 3 (HTTP/3),” Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-27, Feb. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-27

[25] C. B. Krasic, M. Bishop, and A. Frindell, “QPACK: Header Compression for HTTP/3,” Internet Engineering Task Force, Internet-Draft draft-ietf-quic-qpack-14, Feb. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-qpack-14

[26] J. Leitão, P. Á. Costa, M. C. Gomes, and N. M. Preguiça, “Towards enabling novel edge-enabled applications,” Tech. Rep., 2018. [Online]. Available: http://arxiv.org/abs/1805.06989