Recent Progress on the QUIC Protocol

Mehdi Yosofie, Benedikt Jaeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: mehdi.yosofie@tum.de, jaeger@net.in.tum.de

***Abstract*—Internet services increase rapidly and much data is sent back and forth inside it. The most widely used network infrastructure is the HTTPS stack which has several disadvantages. To reduce handshake latency in network traffic, Google's researchers built a new multi-layer transfer protocol called Quick UDP Internet Connections (QUIC). It is implemented and tested on Google's servers and clients and proves its suitability in everyday Internet traffic. QUIC's new paradigm integrates the security and transport layer of the widely used HTTPS stack into one and violates the OSI model. QUIC takes advantages of existing protocols and integrates them in a new transport protocol providing less latency, more data flow on wire, and better deployability. QUIC removes head-of-line blocking and provides a pluggable Congestion Control interface.**

This paper indicates the disadvantages of the traditional HTTPS stack and presents the main features of the QUIC protocol which is currently standardized by the Internet Engineering Task Force (*IETF*).

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

1. Introduction

The Internet is used every day by many readers of this paper. Internet giants like Amazon, Google, and Facebook provide many applications and (web) services which let the amount of data grow significantly. This data has to be exchanged fast, reliably, and securely. Transferring this data is possible through the existing infrastructure. The most widely used one is the HTTPS stack. HTTP is transported over TCP and is secured through TLS. This network paradigm is approved and will be used for many years to come. However, the speed to built up a connection between client and server and to deliver data between them can be improved. Quick UDP Internet Connections (QUIC) may be a solution and a good replacement of the traditional Internet stack. The new paradigm of QUIC combines the transport layer and the security layer into one and provides improved features than TCP/TLS. QUIC is developed by Google. The researchers implemented the protocol and tested it on production mode on their servers such as YouTube and other Google web services, and client applications such as Chrome/Chromium. To be able to communicate over QUIC, both server and client have to provide a QUIC implementation. Thus, QUIC requires client support on application level like in the browser. QUIC aroused the interest of the Internet Engineering Task Force (*IETF*) and is on standardization progress. The *IETF* is an Internet committee which deals with Internet technologies and publishes Internet standards. Currently, QUIC is being standardized, and it remains to be seen, how it will influence the Internet traffic afterwards.

The rest of this paper is structured as follows: Section 2 presents background information about the established TCP/TLS stack needed for the problem analysis. Section 3 explicitly investigates some QUIC features like stream-multiplexing, security, loss recovery, congestion control, flow control, QUIC's handshake, its data format, and the Multipath extension. They each rely on current *IETF* standardization work, and are compared to the traditional TCP/TLS stack. In Section 4, related work about comparable protocols like SPDY or SCTP, and other work of *IETF* is discussed. Section 5 concludes this paper.

2. Background

The Transmission Control Protocol (TCP) is the standard transport protocol that most applications are based on. Data transport over TCP is reliable and packets are organized in an ordered way. Further, lost data is identified and delivered again. Other TCP features are congestion control and flow control which are necessary to not overload the receiver or the network. All of these concepts are relevant and necessary in the transport layer so that application level protocols like HTTP do not have to care about and it is ensured that application level data are not missing on endpoints. In the traditional web stack, TCP is used as the underlying protocol to transport HTTP data. To refer to the OSI model, transferring HTTP over TCP is additionally protected by TLS. TLS is a security setup which is on top of TCP (compare Figure 1). Although it is the widely used transport protocol to transmit data reliably, TCP has several disadvantages.

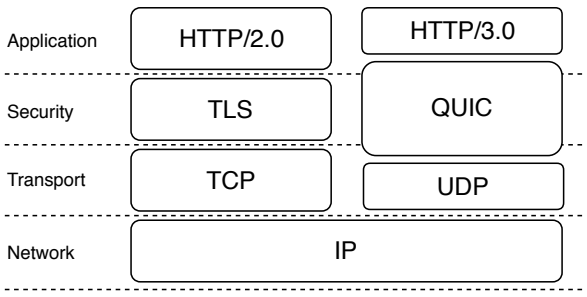

Figure 1: Traditional network stack and QUIC in comparison, adapted from [1]

doi: 10.2313/NET-2019-06-1_16

It is more difficult to have a faster development cycle and publish new releases of kernel based implementations like TCP. This protocol is implemented as part of the kernel in an operating system, i.e. network connections based on TCP run in kernel mode. Devices often have to be upgraded on both client and server side. Bringing changes to TCP and thus to the kernel would in turn cause changes to the operating system. Moreover, TCP has some inconveniences like the head-of-line-blocking delay and the handshake delay. To transfer data, at least one round trip is needed to set up the TCP connection. Furthermore, the security layer with TLS adds two further round trip delays on top TCP connections (compare Figure 1). Even if the handshake delay seems to be solved with TLS 1.3 and TCP Fast Open, the data transfer can still be optimized.

To reduce handshake latency, there is a new approach. QUIC runs – in contrast to TCP – in user space, and uses the User Datagram Protocol (UDP) as the underlying transport protocol. UDP is a widely used and lightweight transport protocol. Thus, it is suitable to be used as transport layer protocol to transmit data from host to host. The advantage of UDP is that it can traverse middleboxes like firewalls. Many firewalls, especially in big companies, block unknown protocols. Thus, unfamiliar packets could be dropped by middleboxes. QUIC encrypts and authenticates its packets and makes it possible to be transported by UDP. There is a higher probability to get UDP packets through the Internet. However, because of the required, but missing TCP features in UDP, QUIC has to implement everything that makes the protocol safe, secure, fast, and reliable by its own: QUIC has to implement congestion control and flow control. It has to define its handshake, and, concerning security reasons, it also should decide for an algorithm about encryption and authentication. These features are implemented by QUIC on the application layer. Additionally, a reason why TCP is not used, is its slow development cycle. This makes it easier to decide for the data transport protocol UDP. Thus, it is more comfortable to bring new releases and adapt features of QUIC without concerning the long term development cycle of the kernel based TCP.

TCP's handshake is the well known *Three-Way-Handshake* (RFC 793). It takes one round trip until data can be sent. If the current overall used TLS version, TLS 1.2 is used on top of TCP for encryption and authentication, two more round trip delays come additionally according to RFC 5246. This would sum up to three round trips until payload can be sent. For the QUIC standardization, the new version 1.3 of TLS will be used. Because QUIC puts encryption and transport together into the same layer in user space, it is possible to overlay the key exchanges of encryption and transport, and have a 0-RTT handshake to same servers. TLS 1.3 is already an *IETF* standard described in RFC 8446. However, hitherto this protocol is barely implemented, however, it will be the standard security protocol from the near future on. Ubuntu announced in its new interim release 18.10 the updated OpenSSL package which is equipped with TLS 1.3. Some applications use TLS 1.3 as default on the new OS version. According to [2], in the next Ubuntu release, TLS 1.3 will be used by more applications.

Users could firstly use QUIC through Google applications like the Chrome browser, the open source version Chromium, and the YouTube application on Android [1]. Gradually, there are more and more both client and server side supports. The Opera browser also supports QUIC if the corresponding QUIC flag is enabled. Except that YouTube and all other web based Google services have server side QUIC support, there are other implementations in Go and C/C++ [3], [4] which are partly used in the Caddy web server, and the LiteSpeed web server respectively. An implementation in Rust is also available [5]. Like every other network protocol, QUIC has to be implemented on both server and client to interact with each other. Other browsers and (web) servers could begin to support QUIC after the standardization is finished by the *IETF*. Since Google controls both client and server side of applications, it can implement and deploy such protocols and test it on their servers and world wide used client applications. Google's leading position in the network technology supports them to develop such protocols.

If the QUIC flag is enabled in Chrome and a client does a request via TCP and TLS, the QUIC server advertises with a QUIC flag in his HTTP answer. The next client side request, if the client wishes, will be a race between TCP/TLS and QUIC [1]. The faster reply will be the protocol stack which will be used for that request. QUIC will only be chosen if in the whole path from client to server QUIC is enabled and supported. Chrome and Chromium users can currently set the corresponding QUIC flag in their browsers to activate QUIC. Servers and companies with production based services may disable UDP inputs by their firewalls due to UDP spoofing attacks. In those intranets, QUIC can currently not be used.

3. Standardization

This Section describes the most essential QUIC features such as security, loss detection, congestion control, flow control, QUIC's data format, its handshake, and the Multipath extension. While QUIC was developed as a monolithic infrastructure by Google [1], the *IETF* work is modularizing it into separate parts. Some details such as the data format will be changed, although the core and the paradigm of QUIC will be the same as initiated by Google. Most of the concepts explained in the next Subsections, are planned to be standalone RFCs [6].

3.1. Stream Multiplexing

To fetch data fast and in parallel, HTTP/1.1 opens multiple TCP connections (compare Figure 2a). However, since each single connection has to be handled, this approach may be inefficient and may cost high CPU rates on constrained devices. HTTP/2.0 proposed to use a single TCP connection, but multiple streams. On every stream, data can be delivered (Figure 2b). The problem of this approach, due to TCP's in order delivery, is the head-of-line-blocking delay. If a packet of one stream is lost, then all other streams are blocked. The head of the line is blocking the whole connection. QUIC allows multiple streams like HTTP/2.0 but does not block all other streams due to a blocking stream (Figure 2c). Data delivery on other streams are not postponed because UDP is not bound on the in-order delivery. According to the latest core protocol draft, streams are identified by a unique stream

doi: 10.2313/NET-2019-06-1_16

ID and data delivery ordering is handled with stream offsets within a stream.

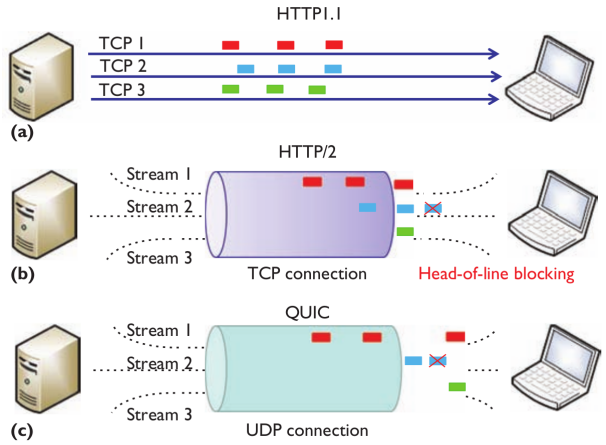

Figure 2: The different HTTP versions and the upcoming HTTP/3.0 in comparison [7]

3.2. Security

Encrypted packets play a key role in Internet traffic. Thereby, network attacks like packet sniffing or man-in-the-middle-attacks can be reduced or even stopped. Only authorized entities should be able to read specific packets in their original formats. This means, QUIC as a secure transport protocol has to provide confidentiality, integrity and authentication.

The first launch of the QUIC development was around 2012 by Google. The researchers implemented their own cryptography for QUIC to reach the goal to exchange payload with zero round trip delay. Since TLS 1.3 is deployed as standard, the *IETF* working group decided for TLS 1.3 as the security layer for QUIC. TLS 1.3 provides a 0-RTT handshake to known servers and thus fits to QUIC's target to reduce handshake lateny.

According to the current Chromium testable implementation, a QUIC server needs to have a valid certificate, and a private key in correctly supported formats to run [8]. TLS is integrated into QUIC handshakes. The current document draft about QUIC-TLS mentions that a server must have a certificate signed by a valid certificate authority, and the client must authenticate the identity of the server during the handshakes. This would mean that even a test server has to install a certificate to be runnable. The positive effect would be a tendency to the overall HTTPS usage and thus, to secure and authenticate web servers in the Internet.

3.3. Loss Recovery

QUIC packets always have increasing packet numbers and same packet numbers do not occur in a number space. This concludes that packets with higher packet numbers are sent later than packets with lower packet numbers. This way, there will not be the retransmission ambiguity problem as in TCP. In TCP, if a packet is sent, and no ACK is received within a timer, the same packet is sent again with the same sequence number. If an ACK arrives, it cannot be determined, which packet is acknowledged, and the round trip time is not measured reliably. With monotonically increasing sequence numbers, the RTT can be determined more accurately and there is no ambiguity problem. Further, regarding loss recovery, a packet which is declared lost, will have a new sequence number and is sent again [9].

3.4. Congestion Control

To reduce and avoid high network load when too many packets are sent by many endpoints in short times and packets cannot pass on, there is the need of congestion control algorithms to control the network rate and reduce the network load. The Google implementation of QUIC is adjustable so that any congestion control algorithm can be experimented and may work. During the development and testing cycle, Cubic was used as congestion controller by the Google developers [1]. Cubic is also the standard congestion controller of the Linux TCP [10]. The first *IETF* draft concerning congestion control in QUIC also mentions that Cubic was the default congestion controller. At that time, Reno was another option to use, since it is fully implemented [9]. In the latest draft of the *IETF* working group about congestion control, NewReno is announced on what QUIC bases on. However, it is also emphasized that every host and every implementation may use a different congestion controller, since QUIC supports a pluggable congestion control interface.

3.5. Flow Control

Flow control is a data transport feature not to overload the receiver. If an endpoint cannot receive data as fast as the sender sends, the receiver is overstrained, cannot handle all incoming packets and drops packets. Flow control mechanism handles the data flow between the sender and receiver so that the receiver does not get more data than its buffer can store. QUIC has, similiar to HTTP/2, two levels of flow control: stream-level flow control and connection-level flow control. Stream flow control limits the data flow for every single stream so that a specific stream will not be able to claim the entire receive buffer for itself and to preclude other streams which send data on other streams. Connection flow control means, the sender does not exceed the receiver's buffer on a connection for all streams. However, it works the same way as stream flow control, for the entire connection.

3.6. Handshake

QUIC distinguishes between the 1-RTT and the 0-RTT handshake. The cryptographic handshake minimizes handshake latency by using known server credentials on repeat connections. A client stores information about the server and can use the 0-RTT handshake on subsequent connections to the same origin. The 1-RTT handshake is possible because the transport and cryptographic keys are overlapped into the same layer. The 0-RTT handshake is possible, since TLS 1.3 provides a 0-RTT and TLS 1.3 will be used as security layer in QUIC. Since many of the network connections are to same servers which

doi: 10.2313/NET-2019-06-1_16

were contacted before, the 0-RTT makes it possible for a client to send payload without repeating cryptographic or transport key exchange.

3.7. Data Format

The data format and the naming conventions of QUIC packets and its fields are repeatedly in change during the standardization. The current draft distinguishes between a long header packet and a short header packet. The long header packet has different types: *Initial*, *Retry*, *Handshake*, and *0-RTT Protected*. Thus, the initial connection between two communication partners is established using the long header. After connection establishment, the short header is used. All types of packets ensure confidentiality and integrity [11].

Furthermore, according to the latest draft, there is the *Version Negotiation Packet* which seems to be a long header packet when received by clients; but as the Version Field is 0, that packet is recognized as a *Version Negotiation Packet*. This packet is only sent by servers and is an indication of the server to a client which versions are supported by that QUIC server. All supported QUIC versions on the server are listed in that packet. It will be sent after the server received a packet with a version proposal that it does not support. If the client side chosen version is not supported by the server, the server has to send a *Version Negotiation Packet* which results in one additional round trip delay before the connection is established [11].

3.8. Multipath Extension

A further feature of the QUIC protocol is *Multipath*. A QUIC flow is, in contrast to a TCP connection, not bound to the 5-tuple consisting of source IP/Port, destination IP/Port, and the transport protocol itself. Instead, the protocol specifies a *Connection ID* each one for the server and the client which is placed in every QUIC packet header. Users can switch from one to another network seamlessly and still communicate with the same server. Using multiple paths over the Internet, with changed values in the 4-tuple, is specified through the QUIC Migration feature. A QUIC implementation including the *Multipath* extension in Go can be found in [3].

4. Related Work

HTTP/1.0 was standardized by the *IETF* in 1996 in RFC 1945. Three years later, the *IETF* introduced the second version HTTP/1.1 which is described in a stricter way with clearer rules in RFC 2616. Companies like Google and Microsoft always want the Internet to be faster. Google initiated the SPDY project [12] and on the basis of SPDY, the new version of HTTP was standardized by *IETF*: HTTP/2.0 was announced in 2015 (RFC 7540). It uses the multiplexing technology which transfers more data. On the incoming side, data is demultiplexed again. Among other reasons, and because of that, HTTP/2.0 is faster than HTTP/1.1. QUIC is the continuous development of Google's research to reduce latency in the web and transmit data faster. It is the successor of SPDY and the standardized HTTP/2.0 protocol.

The *IETF* currently has a working group for the transport protocol QUIC. Beside the core overview, each essential feature of QUIC is described into separate drafts which are planned to be standalone RFCs. There is also a working group responsible for "HTTP over QUIC" which describes the transport of HTTP over QUIC as transport layer. The researchers recently discussed about naming conventions and decided to rename "HTTP over QUIC" to "HTTP/3" [13]. HTTP version 3 would mean that the HTTP protocol would run on the base of the QUIC Transport Protocol. This would be another milestone for the future of the Internet.

To reduce handshake latency in TCP, there were some improvements. TCP Fast Open (TFO) allows to send data in the TCP SYN field to same servers. Thus, after connection establishment, there is a 0-RTT to send payload. But data can only be sent as much as the TCP SYN segment offers. QUIC does not have this limitation. Only the congestion controller or the flow controller can limit the data which can be sent in 0-RTT handshakes [1].

Another affiliated aspect in relation to QUIC is the Stream Control Transmission Protocol (SCTP) defined in RFC 4960. SCTP has many similarities and parallels to TCP. It is a connection oriented transport protocol which also uses sequence numbers and acknowledgments to provide reliable data delivery. Like TCP, SCTP uses a window mechanism to signal how much data can be delivered on receive buffers. Even if the terminology of SCTP is quite different, they both share common features. However, SCTP was built to introduce some advantages over TCP. It is not bound to a single IP address, not even on the IP versions. Moreover, both hosts of a SCTP connection can have multiple IP addresses to communicate with each other. SCTP transmits on multiple streams and is not bound on delivering data in order [14]. There is no head-of-line-blocking. The main problem of SCTP is that is not widely deployed. While the Linux kernel implemented SCTP, MacOS and Windows do not support SCTP officially, but only through extensions. Thus, SCTP is not spread widely. QUIC takes the main idea of SCTP and introduces stream-multiplexing without head-of-line-blocking. It provides *Multihoming* through the *Multipath* extension. However, through the aggregation of the transport and security protocols and the usage of UDP, handshake latency is reduced by QUIC.

5. Conclusion and Future Work

Latency Reduction was a key note when QUIC was developed by Google. Simultaneously, other transport features within the Internet such as security and reliability had to remain unchanged. Google reached these goals by deploying QUIC and tested it in production mode within Chrome/Chromium on YouTube and other Google services. The underlying UDP as transport layer is a new approach to deliver data still reliably. This leads to implement additionally separate features like congestion control, flow control, and loss recovery, but on another level and on top of UDP. That causes QUIC to run in user space. Indeed, features like latency improvements and removing head-of-line-blocking make QUIC attractive and let QUIC be a proposed standard for today's Internet.

doi: 10.2313/NET-2019-06-1_16

However, the protocol violates the OSI model. The transport features and the security protocol are mapped into the same layer in the application level without following the widely established OSI reference model. Thus, it remains to be seen which influence on the usage of the traditional TCP/TLS infrastructure will be. It also remains to be seen how the standardization process by *IETF* evolves and when the standard will be announced. Following the *IETF* milestones of the QUIC working group, most of the drafts' due dates were changed from November 2018 to July 2019 and one to May 2020. The *"Core Protocol document"* of QUIC is planned to be finished in July 2019 [6].

References

[1] A. Langley *et al.*, "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

[2] D. J. Ledkov, J.-B. Lallement *et al.*, last visited 14 February 2019. [Online]. Available: https://wiki.ubuntu.com/CosmicCuttlefish/ReleaseNotes

[3] L. Clemente, M. Seemann *et al.*, "lucas-clemente/quic-go," Feb 2019, last visited 16 February 2019. [Online]. Available: https://github.com/lucas-clemente/quic-go

[4] D. Tikhonov *et al.*, "litespeedtech/lsquic-client," Feb 2019, last visited 16 February 2019. [Online]. Available: https://github.com/litespeedtech/lsquic-client

[5] B. Saunders, D. Ochtman *et al.*, "djc/quinn," Feb 2019, last visited 16 February 2019. [Online]. Available: https://github.com/djc/quinn

[6] "QUIC (quic)," last visited 17 February 2019. [Online]. Available: https://datatracker.ietf.org/wg/quic/about/

[7] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kühlewind, "Innovating transport with QUIC: Design approaches and research challenges," *IEEE Internet Computing*, vol. 21, no. 2, pp. 72–76, 2017.

[8] "Playing with QUIC," last visited 17 February 2019. [Online]. Available: https://www.chromium.org/quic/playing-with-quic

[9] [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-quic-recovery

[10] "SNMP counter," last visited 14 February 2019. [Online]. Available: https://www.kernel.org/doc/html/latest/networking/snmp_counter.html

[11] "QUIC: A UDP-Based Multiplexed and Secure Transport." [Online]. Available: https://tools.ietf.org/html/draft-ietf-quic-transport-16

[12] "SPDY: An experimental protocol for a faster web," last visited 14 February 2019. [Online]. Available: https://dev.chromium.org/spdy/spdy-whitepaper

[13] M. Nottingham *et al.*, "IETF Mail Archive," last visited 14 February 2019. [Online]. Available: https://mailarchive.ietf.org/arch/msg/quic/RLRs4nB1lwFCZ_7k0iuz0ZBa35s

[14] J. R. Iyengar, P. D. Amer, and R. Stewart, "Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths," *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 951–964, Oct 2006.