Innovating Transport with QUIC: Design Approaches and Research Challenges

Yong Cui, Tianxiang Li, and Cong Liu • *Tsinghua University, China*

Xingwei Wang • *Northeastern University, China*

Mirja Kühlewind • *ETH Zurich, Switzerland*

QUIC UDP Internet Connections (QUIC) is a new transport protocol that provides low-latency communication, security, and rapid deployment. QUIC has begun its standardization process with strong interest in the IETF community. This article introduces QUIC's features and discusses its challenges.

Networks these days need to handle a lot more connections, with a growing demand for low latency without sacrificing security and reliability. However, applications are often limited by the use of TCP as the underlying transport. TCP, without the TCP Fast Open extension, introduces one round-trip time (RTT) of latency due to its handshake. It might slow down performance when packet loss occurs and when packets are retransmitted with long delays leading to head-of-line blocking. Moreover, TCP is built in the system kernel, which brings difficulties to the protocol update.

To tackle these issues, a new transport protocol called Quick UDP Internet Connections (QUIC) has been proposed. QUIC is defined on top of UDP, and its design is inspired by the best practices of multiple existing protocols, including TCP, Transport Layer Security (TLS), and HTTP/2. QUIC aims to reduce connection latency by sending data directly when establishing a connection in the best case (the so-called "0-RTT" approach). Furthermore, it provides multiplexing features optimized for HTTP/2 and richer feedback information that might allow for new congestion control approaches. Moreover, as encapsulated in UDP, QUIC can be easily implemented in user space instead of the system kernel, which enables faster deployment as part of application update cycles.

The QUIC protocol is currently being standardized by the IETF QUIC working group. The IETF community showed strong interest in standardization of QUIC. A previous version has been deployed in most Google services as well as the Chrome browser, and is being implemented by a few third-party developers. It should be noted that the standardization process of QUIC is fully open to community input that might lead to significant differences in the protocol design, as compared to the version currently deployed by Google.

Design Overview

Figure 1 shows the layering approach of QUIC. QUIC incorporates congestion control and loss recovery features similar to TCP, while providing richer signaling capabilities. Additionally, QUIC decreases network latency by offering fewer RTTs for connection setup. QUIC incorporates the key negotiation features of TLS 1.3, requiring all connections to be encrypted. The motivation behind mandatory encryption isn't just to ensure security and privacy of user data, but also to prevent middle boxes from tampering with the packet information, which can hinder the future evolution of the QUIC protocol. Further, QUIC also subsumes features of HTTP/2 such as multi-streaming, while avoiding problems such as head-of-line blocking that occur when TCP is used, because all packets (of potentially different HTTP/2 streams) must be delivered in order.

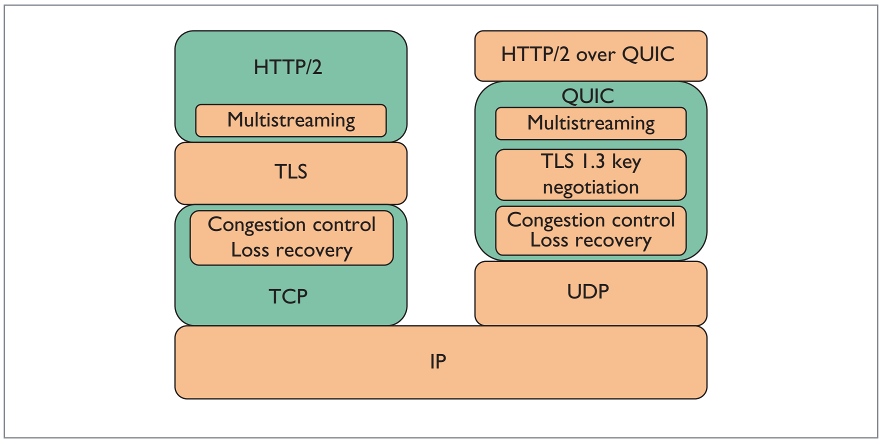

Figure 1. Quick UDP Internet Connections (QUIC) architecture. TLS = Transport Layer Security.

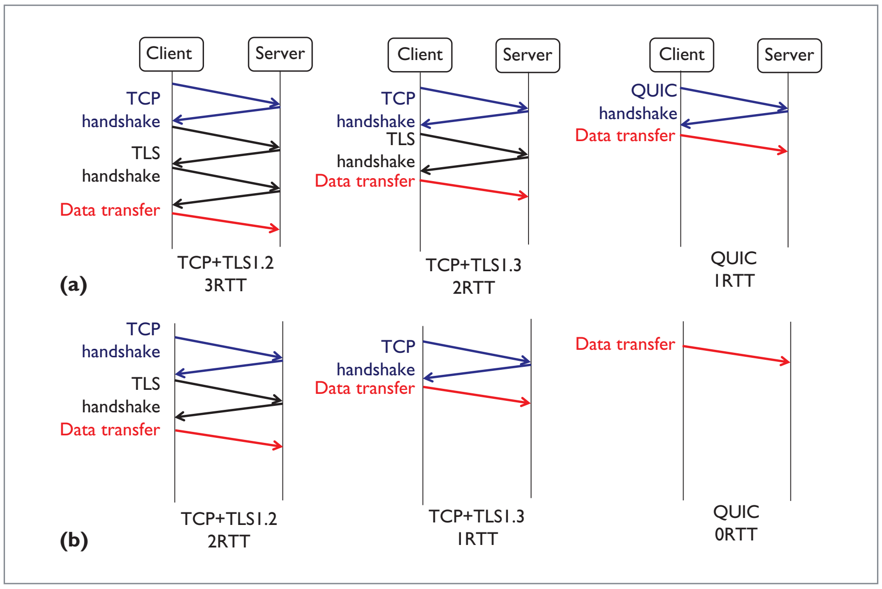

Figure 2. Handshake round-trip time (RTT) of different protocols. (a) First-time connection establishment. (b) Subsequent connections.

Connection Establishment

A majority of services these days require a secure and reliable network connection, and TCP+TLS are widely used for fitting this purpose. One issue with TCP+TLS (1.2) is that it takes at least two RTTs to set up a secure connection, which brings a significant latency overhead. QUIC improves on this by tightly integrating with TLS1.3, leading to a minimum of zero RTTs to establish an encrypted connection — meaning that payload data can be sent on the first packet if a previous encryption session is resumed.

First-Time Connection Establishment

With successful version negotiation, QUIC uses one RTT for the first-time connection establishment by combining the transport and crypto handshake, which is two RTTs less than the widely used TCP+TLS 1.2 and one RTT less than TCP+TLS 1.3 (see Figure 2a). QUIC combines the transport and crypto handshake to minimize connection latency, carrying both the TLS handshake and the relevant QUIC transport setup parameters in the first packet of the connection.

When the QUIC client is connecting to a server for the first time, it sends the Client Hello message to the server for key negotiation, along with some basic QUIC options and parameters such as the connection identifier as well as the preferred version number. The client encodes the handshake according to the version number it proposed. If the server doesn't support the version, it would trigger the client to go through an additional version negotiation process. Otherwise, the server replies with the Server Hello message, certificate, and session information that the client can use the next time it connects to the server. The client can then send its encrypted requests to the server, taking a total of one RTT for connection setup.

The parameters negotiated during the first connection are contained in a cryptographic cookie stored on the client. It's used to authenticate the client when the client connects to the same server again. The cookie also contains the server's Diffie-Hellman value, which is used to calculate the encryption key. This information is the basis for the 0-RTT connection establishment.

0-RTT Connection Establishment

Many connections are established between clients and servers that had communicated before, making it possible to reduce negotiation latency if the server could recognize the client during subsequent connections. Not requiring one RTT for the transport handshake and utilizing session resumption for encryption allows a

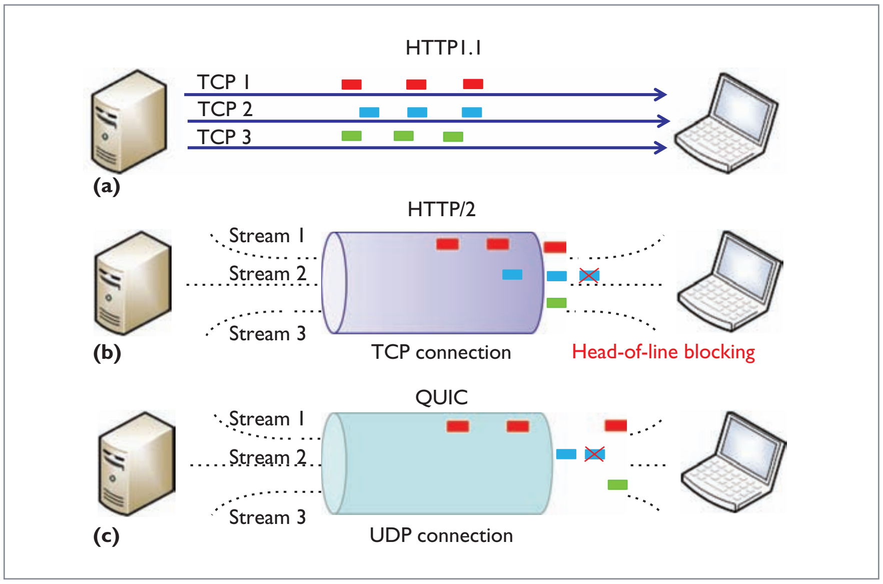

Figure 3. Multiplexing comparison. This involved sending multiple streams of data over a single transport connection using (a) HTTP1.1, (b) HTTP/2, and (c) QUIC.

QUIC client to immediately send data to the server it had connected to before (see Figure 2b).

To resume a cryptographic session, the client sends its cached cryptographic cookie and Diffie-Hellman value to the server along with the encrypted payload. The server authenticates the client through information contained in the cookie. After successful authentication, it can calculate the encryption key using the Diffie-Hellman value stored in the cookie and the value sent by the client. The server can then decrypt the payload data — such as a HTTP/2 request — and send its encrypted response immediately back to the client.

Stream Multiplexing

Stream multiplexing is a method for sending multiple streams of data over a single transport connection. Browsers usually open multiple concurrent TCP connections when accessing a website because HTTP1.1 could only request one resource at a time (see Figure 3a), which consisted of short data transfers over independent connections. This introduced additional latencies and the complexity of managing multiple connections.

HTTP/2 addresses this problem by multiplexing streams over one TCP connection if multiple requests are sent to the same server. However, even if the payload data of the different streams are independent, all data transmitted over the same TCP connection will be delivered in order to the application, leading to head-of-line blocking of missing data on one stream for data successfully transmitted and received on other streams, as Figure 3b shows.

QUIC supports multiplexing of concurrent HTTP streams on a single connection without requiring ordered delivery of all packets of the transport connection, as Figure 3c shows. In QUIC, all data are still transmitted fully reliably. One QUIC packet can carry multiple frames of the same or different streams. All the frames belonging to the same stream are delivered in order, but the missing frames of one stream don't block the delivery of other streams' payload data.

QUIC also adapts two levels of flow control similar to HTTP/2 over TCP. Connection-level flow control allows adjustment of the aggregate buffer for the entire connection. Stream-level flow control allows the receiver to adjust how much data it's willing to allocate for each stream, avoiding that a single stream consumes all the buffer resources and thus could block other stream transmissions.

Congestion Control and Loss Recovery

The QUIC working group in the IETF is currently chartered to only use standardized congestion control as the default congestion control algorithm, which at the moment is just NewReno[1] and Cubic.[2] Cubic is a widely used congestion control mechanism and is undergoing activity in the IETF TCPM working group. Similar to most TCP implementations today, QUIC aims for a pluggable congestion control interface that allows experimentation with different congestion control algorithms.

However, QUIC provides a slightly different environment for congestion control than TCP does. First, it inherently adopts modern loss-recovery mechanisms such as F-RTO[3] and Early Retransmit.[4] Further, it offers more detailed feedback information for loss detection. For example, it uses a monotonically increasing packet number but doesn't retransmit on the packet-level (only on a per-frame base). This allows QUIC to distinguish retransmissions from the originally sent packets, avoiding retransmission ambiguities, similar to the idea of the TCP Recent ACKnowledgment (RACK) algorithm.[5] Additionally, QUIC carries information about the delay between when a packet was received and when the ACK was sent. This information allows the original sender to achieve a better estimation of the path RTT. QUIC also adopts the TCP's selective acknowledgment mechanism,[6] and supports up to 255 ACK ranges, making it more resilient to reordering and loss.

Challenges and Future Directions

The first QUIC working group meeting was held at IETF-97 and the initial working group documents have been adopted shortly after, focusing on the design of the core transport protocol;[7] the congestion control and loss recovery mechanism;[8] using TLS 1.3 for key negotiation;[9] and a mapping for HTTP/2.[10] The QUIC working group charter foresees multipath support and optional forward-error correction (FEC) as the next step, but are currently out of scope until all action items on the current milestone list are completed. Further, the working group also focuses on network management issues that QUIC may introduce, aiming to produce an applicability and manageability statement in parallel to the actual protocol work. Aside from this work, QUIC also provides the potential for research in other areas.

Congestion Control in Special Network Scenarios

Congestion control mechanisms in transport protocols need to adapt to many different scenarios with distinct, varying network characteristics. Here, wireless networks are a key challenge for congestion-control research. Traditionally, TCP congestion control regards packet loss as congestion occurrence. For wireless networks, however, loss often indicates transmission errors, especially in the case of mobility of the connecting client. The more fine-grained information provided by QUIC can help to distinguish other loss events from congestion events and therefore a large performance improvement could be achieved in situations with high non-congestion related loss rates.

Other specific network scenarios, such as data centers that require low latency for short flows[11] and virtual reality (VR) that has high demand for user-perceived latency, might also benefit from the more accurate timing information provided by QUIC's feedback scheme. Furthermore, customizing QUIC to specific network scenarios could be achieved more easily in user space implementations of QUIC, which provides a platform for easier experimentation and faster deployment.

Forward-Error Correction

Packet loss leads to congestion window reduction and thereby decreases the throughput, regardless of whether the loss is congestion-related. However, even if congestion occurred and the sending rate is correctly reduced, packet loss still causes additional delays due to potentially slow recovery mechanisms, based on either duplicate acknowledgments or even retransmission timeouts (for example, if tail loss occurs). Coding is a method of using redundant information sent with packets for FEC, providing better loss tolerance and proactive, faster recovery.

One of the issues of coding is the additional time introduced for encoding and decoding, which is against QUIC's low-latency design principle. Further, as the amount of redundant information affects the performance of coding-based loss recovery, it's a tradeoff with bandwidth consumption. Coding could be used to improve performance for scenarios with high packet loss such as wireless networks, but it would also introduce more energy consumption, causing issues for power-constrained mobile terminals. While coding has been well-deployed in the link layer, it's still a point of research for transport layer protocols such as QUIC and TCP.[12]

Application-Based Optimization

Currently, the working group will focus on providing a mapping of HTTP/2 to QUIC as the initial use case. However, QUIC can also be used for other applications. Especially as future versions of QUIC might incorporate FEC, it could be applied to applications such as real-time communication and video streaming, which are tolerant to loss but latency-sensitive. The performance of QUIC could thus be optimized based on application requirements, which requires an interface for the application to configure — for example, the content type and tolerance of packet loss.

Prioritization

Web content usually has dependencies between the Web objects, which might limit performance.[13] For example, a JavaScript file should be loaded prior to a file triggered by the script. Given that QUIC provides multiple, independent streams to transmit these Web objects, it's possible for QUIC to prioritize between streams based on these dependencies. However, setting the priority levels correctly, considering dynamic object load time and current network status, is an open field for additional research.

Security and Privacy

QUIC provides secure transport by integrating the security functionality of TLS and enforcing the encryption of all connection data. However, similar to TLS1.3, 0-RTT resumption in QUIC might also introduce new security threats. A typical kind of issue is the replay or manipulation of packets from a previous connection handshake. Furthermore, if such an attack causes the client and server to perform a full handshake, consuming computational resources and memory space, it could be used as an additional DoS attack vector.[14] Further analysis and research is valuable in this space.

Unencrypted information such as a connection identifier is susceptible to the threat of pervasive monitoring attacks. However, some information is needed for economically viable network management supporting the current common practice of firewalls, load balancers, and network address translation (NAT) traversal. To conserve privacy while allowing for functions such as IP address mobility, it's suggested that QUIC use new identifiers for each encrypted communication session to avoid linkability.[15] This tussle is the subject of an ongoing discussion in the QUIC working group and will be even more relevant when work is extended to include multipath support for QUIC.

QUIC is a new transport protocol currently under standardization in the IETF, introducing features such as 0-RTT connection establishment, stream multiplexing avoiding (packet-based) head-of-line blocking, and improved signaling for loss recovery and congestion control. Given these changed characteristics compared to TCP, QUIC provides new challenges and opportunities for research. Moreover, as QUIC is based on top of UDP, it provides a platform for easy experimentation, and potentially fast adoption and deployment of research results.

Acknowledgments

This work is partly supported by the National Natural Science Foundation of China (grants 61422206, 61120106008, 61225012, and 61572123), the National High Technology Research and Development Program ("863" Program) of China (2015AA016101), and the Tsinghua University Initiative Scientific Research Program (2014Z09103). This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement 688421, and was supported by the Swiss State Secretariat for Education, Research, and Innovation (SERI) under contract 15.0268. The opinions expressed and arguments employed reflect only the authors' views. The European Commission isn't responsible for any use that might be made of that information. Further, the opinions expressed and arguments employed herein don't necessarily reflect the official views of the Swiss government.

References

1. T. Henderson et al., *The NewReno Modification to TCP's Fast Recovery Algorithm*, RFC 6582, 2012.
2. X. Rhee et al., *CUBIC for Fast Long-Distance Networks, draft-ietf-tcpm-cubic-03*, IETF Internet draft, 2 Dec. 2016; https://tools.ietf.org/html/draft-ietf-tcpm-cubic-03.
3. P. Sarolahti, M. Kojo, and K. Raatikainen, "F-RTO: An Enhanced Recovery Algorithm for TCP Retransmission Timeouts," *ACM Sigcomm Computer Comm. Rev.*, vol. 33, no. 2, 2003, pp. 51–63.
4. M. Allman et al., *Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)*, RFC 5827, 2010.
5. C. Cheng, *RACK: A Time-Based Fast Loss Detection Algorithm for TCP, draft-cheng-tcpm-rack-01*, IETF Internet draft, 31 Oct. 2016; https://tools.ietf.org/html/draft-ietf-tcpm-rack-01.
6. M. Mathis et al., *TCP Selective Acknowledgment Options*, RFC 2018, 1996; https://tools.ietf.org/html/rfc2018.
7. R. Hamilton et al., *QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2, hamilton-quic-transport-protocol-01*, IETF Internet draft, 31 Oct. 2016; https://tools.ietf.org/html/draft-hamilton-quic-transport-protocol-01.
8. J. Iyengar, et al., *QUIC Loss Recovery and Congestion Control, draft-iyengar-quic-loss-recovery-01*, IETF Internet draft, 31 Oct. 2016; https://tools.ietf.org/html/draft-iyengar-quic-loss-recovery-01.
9. M. Thomson et al., *Using Transport Layer Security (TLS) to Secure QUIC, draft-thomson-quic-tls-01*, IETF Internet draft, 25 Oct. 2016; https://tools.ietf.org/html/draft-thomson-quic-tls-01.
10. R. Shade et al., *HTTP/2 Semantics Using the QUIC Transport Protocol, draft-shade-quic-http2-mapping-00*, IETF Internet draft, 8 July 2016; https://tools.ietf.org/html/draft-shade-quic-http2-mapping-00.
11. M. Alizadeh et al., "Data Center TCP (DCTCP)," *ACM Sigcomm Computer Comm. Rev.*, vol. 40, no. 4, 2010, pp. 63–74.
12. Y. Cui et al., "End-to-End Coding for TCP," *IEEE Network*, vol. 30, no. 2, 2016, pp. 68–73.
13. X.S. Wang et al., "How Speedy Is SPDY?" *Proc. 11th Usenix Symp. Networked Systems Design and Implementation*, 2014; www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-wang_xiao_sophia.pdf.
14. R.C. Lychev et al., "How Secure and Quick Is QUIC? Provable Security and Performance Analyses," *Proc. 2015 IEEE Symposium on Security and Privacy*, 2015; https://csdl.computer.org/csdl/proceedings/sp/2015/6949/00/6949a214.pdf.
15. W.M. Petullo et al., "MinimaLT: Minimal-Latency Networking through Better Security," *Proc. 2013 ACM SIGSAC Conf. Computer & Comm. Security*, ACM, 2013; https://cr.yp.to/tcpip/minimalt-20130522.pdf.

Yong Cui is a full professor in the Department of Computer Science and Technology at Tsinghua University, China. His research interests include computer network architecture and mobile computing. Cui has a PhD in computer science from Tsinghua University. Contact him at cy@csnet1.cs.tsinghua.edu.cn.

Tianxiang Li is a master's student in the Department of Computer Science and Technology at Tsinghua University. His research interests include TCP, DHCP, DNS, and IPv6. Li has a BS in e-commerce engineering from the Beijing University of Posts and Telecommunications. Contact him at litx14@mails.tsinghua.edu.cn.

Cong Liu is a PhD student in the Department of Computer Science and Technology at Tsinghua University. His research interests include IPv4–IPv6 protocol transition and network security. Liu has a BS in computer science and technology from Tsinghua University. Contact him at cong-liu13@mails.tsinghua.edu.cn.

Xingwei Wang is a full professor in the Computer Science Department at Northeastern University, China. His research interests include future network architecture, cloud computing, and cybersecurity. Wang has a PhD in computer science from Northeastern University. Contact him at wangxw@mail.neu.edu.cn.

Mirja Kühlewind is a postdoctoral researcher at ETH Zurich, Switzerland. She is one of the Transport Area Directors at the IETF, a co-chair of the Measurement and Analysis for Protocols Research Group (MAPRG) in the IRTF, and the coordinator of the EU H2020 MAMI (Measurement and Architecture for a Middleboxed Internet) project. Her current research interests include transport protocols, congestion control, and Internet measurements. Kühlewind has a PhD from the University of Stuttgart. Contact her at mirja.kuehlewind@tik.ee.ethz.ch.