Innovating transport with QUIC: Design approaches and research challenges

Innovating Transport with QUIC: Design Approaches and Research Challenges

Yong Cui, Tianxiang Li, Cong Liu • *Tsinghua University, China*
Xingwei Wang • *Northeastern University, China*
Mirja Kühlewind • *ETH Zurich, Switzerland*

In today's Internet, there are many challenges such as low latency support for interactive communication, security and privacy of user data, as well as development and deployment of new transport mechanisms. QUIC is a new transport protocol that addresses these challenges with focus on HTTP/2 transmission as a first use case. The first QUIC working group meeting took place at IETF-97 in November, 2016, and it has begun the standardization process. This article introduces the key features of QUIC and discusses the potential challenges that require further consideration.

Introduction

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 may slow down performance when packet loss occurs and when packets are retransmitted with long delays leading to Head-of-line blocking. Moreover, building a transport over UDP allows user-space implementation, enabling faster protocol evolution.

To tackle these issues, a new transport protocol called 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, TLS, and HTTP/2. QUIC aims to reduce connection latency, by sending data directly when establishing a connection in the best case (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 of the 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 currently version deployed by Google.

Design Overview

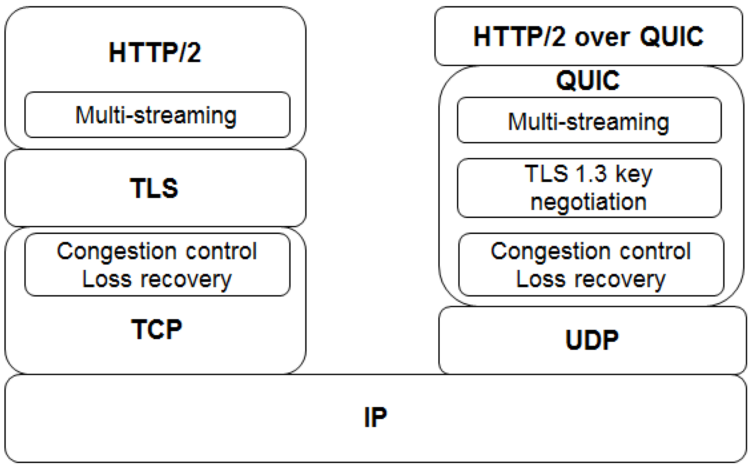

Figure 1. QUIC Architecture

The layering approach of QUIC is shown in Figure 1. 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 set up. QUIC incorporates the key negotiation features of TLS 1.3, requiring all connections to be encrypted. The motivation behind mandatory encryption is not 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 like Head-of-Line blocking that occurs when TCP is used because all packets (of potentially different HTTP/2 streams) have to be delivered in order.

Connection Establishment

A majority of services nowadays require 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.

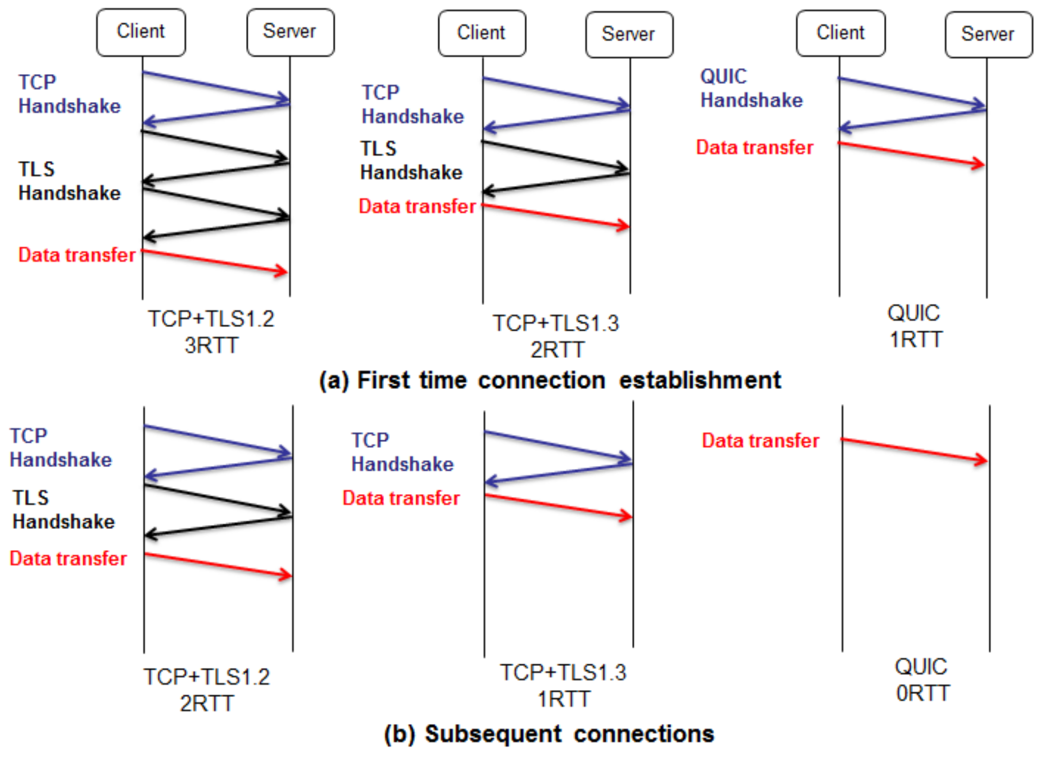

Figure 2. Handshake RTT of different protocols

First-time Connection Establishment

With successful version negotiation, QUIC uses one RTT for the first-time connection establishment by combing 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 as shown in Figure 2 (a). QUIC combines the transport and crypto handshake to minimize connection latency, carrying both the TLS handshake and the relevant QUIC transport set up 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 does not 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 is 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 which 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 QUIC client to immediately send data to the server which it had connected to before, as shown in figure 2 (b).

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, e.g. a HTTP/2 request, and send its encrypted response immediately back to the client.

Stream-Multiplexing

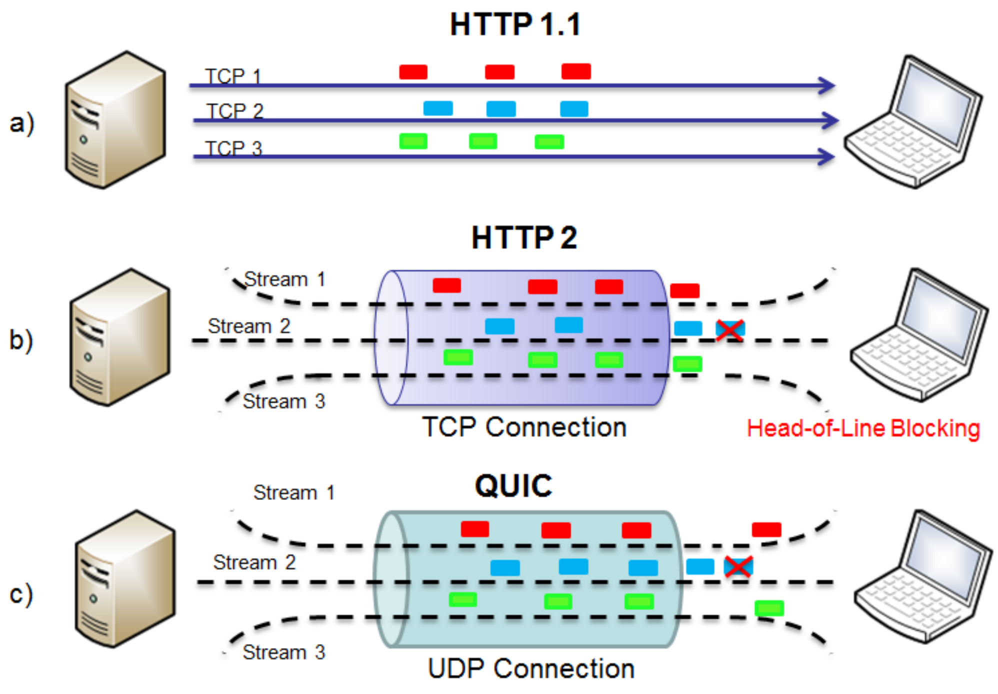

Figure 2. Multiplexing Comparison

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, as shown in figure 3 (a), 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 multiple streams into one TCP connection if multiple requests are sent to the same server. However, even if the payload data of the different stream are independent, all data transmitted over the same TCP connection will be deliver 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 illustrated in figure 3 (b).

QUIC supports multiplexing of concurrent HTTP streams on a single connection without requiring ordered delivery of all packets of the transport connection. In QUIC, all data is 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 other streams do not 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 is 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 is at the moment just NewReno[1] and Cubic[2]. Cubic is a widely used congestion control mechanism and is an under-going activity in the IETF TCPM working group. Similar as most TCP implementation today, QUIC aims for a pluggable congestion control interface which 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[1] and Early Retransmit[4]. Further, it offers more detailed feedback information for loss detection. For example, it uses monotonically increasing packet number but does not 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 TCP RACK[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 acknowledgement mechanism, and supports up to 255 ACK ranges, making it more resilient to re-ordering 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[6], the congestion control and loss recovery mechanism[7], using TLS 1.3 for key negotiation[8], and a mapping for HTTP/2[9]. The QUIC working group charter foresees multipath support and optional forward error correction as the next step but are currently out of scope until all action item of the current milestone list have been 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 these work, QUIC also provides the potential for research in a number of 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 datacenter which requires low latency for short flows[10], and virtual reality (VR) which has high demand for user-perceived latency, might also benefit from the more actuate timing information provided by QUIC's feedback scheme. Furthermore, customizing QUIC to specific network scenarios could be more easily achieved 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, despite of whether the loss is congestion related or not. However, even if congestion occurred and the sending rate is correctly reduced, packet loss still causes additional delay due to potentially slow recovery mechanisms based on either duplicate acknowledgements or even retransmission timeouts, e.g. if tail lost occurs. Coding is a method of using redundant information sent with packets for forward error correction (FEC), providing better loss tolerance and pro-active, 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 is a trade off 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 is still a point of research for transport layer protocols like QUIC and TCP[11].

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, video streaming, which are tolerant to loss but latency-sensitive. The performance of QUIC may thus be optimized based on application requirements, which requires an interface for the application to configure, e.g., the content type and tolerance of packet loss.

Prioritization

Web content usually has dependencies between the web objects, which may limit performance[12]. 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 is 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 as TLS1.3, 0-RTT resumption in QUIC may also introduce new security threats. A typical kind of issue is the replay or manipulation of packet 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[13]. Further analysis and research is valuable in this space.

The 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, NAT traversal and such. In order to conserve privacy while allowing for functions such as IP-address mobility, it is suggested that QUIC use new identifiers for each encrypted communication session to avoid linkability[14]. This tussle is the subject of an on-going discussion in the QUIC working group and will be even further relevant when work is extended to include multipath support for QUIC.

Conclusion

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 experiment and potentially fast adoption and deployment of research results.

Acknowledgement

This work is partly supported by the National Natural Science Foundation of China (grants 61422206, 61120106008, 61225012, 61572123), the National High Technology Research and Development Program ("863"Program) of China no.2015AA016101, and Tsinghua University Initiative Scientific Research Program no.2014Z09103. This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 688421, and was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0268. The opinions expressed and arguments employed reflect only the authors' views. The European Commission is not responsible for any use that may be made of that information. Further, the opinions expressed and arguments employed herein do not necessarily reflect the official views of the Swiss Government.

References

1. Henderson, Tom, et al. The NewReno modification to TCP's fast recovery algorithm. No. RFC 6582. 2012.
2. Rhee, Xu, et al. CUBIC for Fast Long-Distance Networks. draft-ietf-tcpm-cubic-03. 2016
3. Sarolahti, Pasi, Markku Kojo, and Kimmo Raatikainen. "F-RTO: an enhanced recovery algorithm for TCP retransmission timeouts." ACM SIGCOMM Computer Communication Review 33.2 (2003): 51-63.

4. Allman, M., et al. Early retransmit for TCP and stream control transmission protocol (SCTP). No. RFC 5827. 2010.
5. Cheng, Cardwell. RACK: a time-based fast loss detection algorithm for TCP. draft-cheng-tcpm-rack-01. 2016
6. Hamilton, R., et al. "QUIC: A UDP-based secure and reliable transport for HTTP/2." IETF, draft-hamilton-quic-transport-protocol-01 (2016).
7. Iyengar, J., et al. "QUIC Loss Recovery And Congestion Control." IETF, draft-iyengar-quic-loss-recovery-01 (2016).
8. Thomson, M., et al. "Using Transport Layer Security (TLS) to Secure QUIC" IETF, draft-thomson-quic-tls-01 (2016).
9. Shade, R., et al. "HTTP/2 Semantics Using The QUIC Transport Protocol." IETF, draft-shade-quic-http2-mapping-00 (2016).
10. Alizadeh, Mohammad, et al. "Data center tcp (dctcp)." ACM SIGCOMM computer communication review. Vol. 40. No. 4. ACM, 2010.
11. Cui, Yong, et al. "End-to-end coding for TCP." IEEE Network 30.2 (2016): 68-73.
12. Wang, Xiao Sophia, et al. "How speedy is SPDY?." 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). 2014.
13. Cr Lychev, Robert, et al. "How secure and quick is QUIC? Provable security and performance analyses." 2015 IEEE Symposium on Security and Privacy. IEEE, 2015.
14. Petullo, W. Michael, et al. "MinimaLT: minimal-latency networking through better security." Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. ACM, 2013.