QUIC: *Setting the Foundation*

Austin Dase
Computer and Information Sciences
Towson University
Towson, United States
adase1@students.towson.edu

Joseph Kamau
Computer and Information Sciences
Towson University
Towson, United States
jkamau2@students.towson.edu

Manoja Koneru
Computer and Information Sciences
Towson University
Towson, United States
mkoner1@students.towson.edu

Myungsik Kim
Computer and Information Sciences
Towson University
Towson, United States
mkim58@students.towson.edu

Terrence Pugh
Computer and Information Sciences
Towson University
Towson, United States
tpugh4@students.towson.edu

Yi-Shan Shir
Computer and Information Sciences
Towson University
Towson, United States
yshir1@students.towson.edu

***Abstract*— For decades the internet has been powered and made available with the use of protocols. The Quick UDP Internet Connection known as QUIC is a protocol created by Google to create a solution for some issues with TCP. With Google being able to control both the browser and some of today's most popular websites, the company is uniquely positioned to deploy and actively test the QUIC protocol. Both TCP and UDP are built on top of IP; Google has implemented QUIC on top of UDP allowing Google to embed QUIC inside of the Google Chrome browser. This paper will dive into how QUIC is a reliable, multiplexed transport over UDP, how it reduces latency if its able to remain encrypted, and how it runs in the user-space. This paper should also provide a solid foundation of the QUIC protocol by discussing its current implementation, concepts, design, performance, and current use.**

I. Introduction

QUIC is a reliable, multiplexed transport over UDP. It is always encrypted and has been designed to help reduce latency. QUIC is open sourced and runs in the user-space allowing it to be updated and deployed quickly. QUIC was developed from the ground up at Google to improve performance with HTTPS and enable rapid deployment of transport mechanisms [2]. Google has deployed QUIC to thousands of servers and clients including the google web browser known as Chrome [2]. Google has reported that as of 2017 approximately 7% of the internet traffic is now using this protocol. QUIC has also incorporated and transferred years of experiences and best practices from TCP to ensure that it maintains best practices that were developed and proven with TCP. This includes TCP's cubic fairness, FACK, TLP, F-RTO and early retransmit [1]. Along with existing TCP best practices, QUIC also adds signaling enhancements that are not possible to perform with TCP like using new sequence numbers during retransmission. QUIC also allows improved congestion feedback and control over acknowledgments.

II. Concepts and design

A. Concepts

To understand QUIC's concepts and design, we must discuss several goals that QUIC strives to achieve. QUIC focuses on deployability, security, reduced handshake and head of line blocking delays [2]. QUIC reduces setup up and Round-trip times (RTT) by merging cryptographic and transport handshakes [2]. Figure 2 illustrates the process to establish a connection using the QUIC protocol. Core concepts for QUIC include Low-latency connections, multiplexing without head-of-line blocking, header encryption, congestion control, connection flow control, connection migration, NAT rebinding resilience, and version negotiation [3].

B. Low Latency

To maintain low latency QUIC combines cryptographic and transport handshakes to allow a secure transport connection. 0-RTT hands are common, and that means that data is immediately sent after the handshake packet, removing the need for waiting for the server to send the client a reply [3]. Multiplexing is a way used to use multiple signals over a shared medium by implementing streams. QUIC ensures that packets that are lost on a stream only affect that specific stream [3]. QUIC constructs packet framing and acknowledgments in a way that enable simple congestion control and loss recovery [3]. One example is that each packet, including retransmission, is assigned a unique sequence number, unlike TCP which uses the same sequence number for retransmission. This results in not needing a mechanism to determine acknowledgments for retransmissions and eliminates head of line blocking [3]. QUIC also automatically detects a high loss connection, and in the case where a connection is identified as a high loss, packets deemed important are automatically sent twice. This reduces the need for retransmission and packet loss detection. Another simple enrichment QUIC implement is the explicit encoding of delays between the recipient of a packet and its acknowledgment being sent. Together this consistently increases the packet number allowing for accurate network RTT [3].

C. Sequence numbers

TCP uses sequence numbers that facilitate reliability and are used to represent the order in which bytes are to be delivered at the receiver. This causes TCP's retransmission ambiguity problem [2]. QUIC's implementation does not have TCP's retransmission ambiguity problem [3]. QUIC carries information with packet framing and acknowledgments to help with congestion control and loss recovery [3]. Between the receipt of a packet and the acknowledgment being sent QUIC's acknowledgments encode the delay [3]. QUIC also uses explicit signaling in acknowledgments to provide accurate round-trip measurements. This feature enables connections to move across IP address changes by a connection IDs that identify connections instead of IP port 5-tuple [2].

D. Flow Control

To perform stream and connection flow control QUIC's receiver advertises the maximum amount of data it will receive per stream [3]. While data is being sent, received, and delivered QUIC's receiver sends MAX_STREAM_DATA frames that increase the advertised limit for that particular stream for flow control. Along with flow control, QUIC also implements connection-level flow control. This is executed to limit the aggregate buffer that QUIC's receiver is willing to allocate to all streams on a connection [3]. This type of flow control works just like stream-level except for the bytes that arrive, and the limits are combined across all streams [3]

E. Identify QUIC Connections Control

to identify QUIC connections the protocol uses 64-bit connection IDs. The protocol utilizes the ID as a constant. This allows connections to survive any changes that might occur with the IP address or port. Other benefits that come from QUIC's connection migration is that it enables the client to regain shared states after moving networks [3]. This also means recovering outstanding requests that would normally be lost [3]. QUIC's connection migration has privacy considerations. With a stable connection ID on multiple network paths allowing passive observers to align activity between paths, a privacy decision needs to be considered [3]. This is whether or not a client that moves between networks wants to allow their activity to be correlated by an entity other than a server. The client can explicitly end linkability between two points of network attachment [3].

F. Version Negotiation

Version Negotiation allows multiple concurrent versions of the protocol deployed. QUIC versions use 32-bit unsigned numbers and 0x00000000 represents version negotiation and 0x00000001 version of QUIC uses TLS [3]. Version negotiations ensure client and server agreement to a specific QUIC version that is mutually supported [3]. The process consists of a client sending a packet to a server, and its size will determine if the server sends a version negotiation packet [3]. Although all clients do not support multiple QUIC versions, those that do shall pad initial packets to show the greatest minimum initial packet size of all considered versions. This makes sure that the server responds if there are mutually supported versions [3]. If the client sends a version that is not acceptable, the server sends a version negotiation packet listing all of the versions that it will accept. With this system, the server is still able to process packets with unsupported versions. The client will send packets until it successfully receives a response from the server or it stops the connection attempt [3].

G. QUIC Frames and Bandwidth

To minimize the use of bandwidth and computational costs, QUIC builds as many frames as possible for each packet. A sender using the QUIC protocol also waits for small periods to bundle frames to send a packet that is not packed. For a packet to be acknowledged, it must have packet protection removed and must have all packets processed [3]. Once this process is completed, a receiver can acknowledge receipt by sending one or more acknowledgment frames. These frames contain a packet number of the received packet. Lost QUIC packets are not retransmitted as a whole, information that can be carried in frames are sent again in new frames as necessary [3]. If application data is sent in STREAM frames, data will be retransmitted in new STREAM frames. If the endpoint has sent an RST_STREAM, it will not be retransmitted [3]. During this process, an ACK frame should contain all unacknowledged acknowledgments [3].

III. Performance and quic at work

QUIC advances TCP in handshake time since QUIC requires 0 to 1 RRT for a handshake, while the handshake time for TLS/TCP includes the time for both TCP and TLS handshakes to complete. As to the handshake latency, when Google tested QUIC in December 2016[2], 88% of QUIC handshakes for desktop users achieve 0-RTT, the remaining connections are those that do not succeed in building connection in 0-RTT and yet complete the handshakes within 2-3RTT. On the other hand, the handshake latency for TCP/TLS increases linearly along with the minimum RTT of the connection.

Along with the 0-RTT, QUIC also performs improved loss-recovery and large default connection throughput. As a result, QUIC on average reduces 8.0% and 3.6% latency of Google Search for desktop users and mobile users. It also reduces the rebuffer rates of Youtube playbacks for desktop users and mobile users by 18.0% and 15.3%[2].

A. Google Search Latency

The reduced Search Latency mainly comes from the low handshake latency. Therefore the Search Latency gains significant differences between QUIC users and TCP/TLS users as the minimum RRT grows for the later.

For global Search Latency, QUIC reduces on average 8.0% for desktop users and 3.6% for mobile users, while the 99th percentile reaches 16.7% and 14.3 reductions [2]. The fact that QUIC connections established by the mobile app achieve 20% less 0-RTT handshake than desktop leads to a lower reduction of Search Latency on mobile devices than desktop devices. Two factors of this reduction stem from the usage and environment of mobile users; in short, the first factor is that, when mobile user switch networks as they move, the IP address changes then invalidates the source-address token cached at the client; secondly, when mobile users switch networks, the data center in service may also change, yet the data center may have a different server configuration than that cached at the client.

The loss recovery mechanisms of QUIC may also reduce Search Latency since TCP retransmission rates tend to increase with minimum RTT while QUIC's loss recovery contributes to better RTT in Search Latency.

B. *YouTube Video Latency*

On average, QUIC connections for video playback achieve 85% 0-RTT handshake for desktop, 65% for YouTube mobile app[2].

For global Video Latency, QUIC reduces 8.0% latency for desktop users and 5.3% for mobile users on average, and the 99th percentile latency for desktop users and mobile users are 10.6% and 7.5% lower than the 99th percentile latency of TCP[2].

C. *YouTube Video Rebuffer Rate*

Unlike Search Latency and YouTube which are sensitive to handshake latency, Rebuffer Rate relies largely on loss-recovery latency and connection throughput. Unlike search requests which are sensitive to handshake latency, video rebuffer rates, as seen on YouTube, rely largely on loss-recovery and connection throughput. The loss-recovery improvements of QUIC strengthen the resiliency to higher data loss rate compared to TCP. Therefore QUIC performs a better Rebuffer Rate for video playbacks[2]. A QUIC client advertises the 15 MB default initial connection-level flow control, which is sufficient in avoiding congestion in flow control. As a result, for playbacks with non-zero rebuffing, videos played at the optimal rates are 2.9% more for desktop and 4.6% more for mobile over QUIC than TCP[2].

IV. Comparing quic with TCP

A. *Speed*

In this section, we can see performance improvements of QUIC over TCP. First, let's see how the application stack differs between TCP and QUIC from the as shown in Figure 1 QUIC is deployed on top of UDP; encryption is handled by QUIC protocol. Since, UDP is a connectionless protocol, all the rationale to assure a reliable connection between a client and a server is taken care of QUIC. Moreover, QUIC is built in the application layer, which implies that any updates to the protocol don't require OS changes. The main performance improvement of QUIC over TCP come from the connection handshake and multiplexing.

a) Connection Handshake: For establishing connection TCP requires a 3-way connection, and, on top of that, TLS connection should also be needed. However, QUIC is built on top of UDP it requires one packet to establish the connection, including TLS. Additionally, if the server knows the client, the zero round trips are needed to establish a secure connection. This reduction in round trips makes web pages load significantly faster.

b) Multiplexing: When Multipath TCP is not used, TCP ports and IP addresses (of both endpoints) are used for identifying the connection. It makes difficult for the client to communicate with a server over multiple ports via a single connection. In contrast, QUIC uses a 64-bit connection identifier randomly selected by the client. Within these connections, multiple streams are used to transport segments. The communication within the client and server is multiplexed, this overcomes the head-of-line blocking issues that are prevalent with TCP connections.

These and several other QUIC features like improved congestion control and forward error correction make improvements on existing TCP technologies [1]. Google has performed extensive testing and measurements, and they found that over QUIC, Youtube video rebuffers are decreased by 15-18%, and Google search latency is decreased by 3.6-8%[1]. These numbers may seem low, but current TCP applications have been highly optimized by Google, so moving to QUIC by even a few percentages is very significant. These improvements have a dramatic impact on the Internet traffic and user experience[1].

B. *Reliability*

The Internet was using TCP primarily because it is a reliable transmission protocol. But where TCP has advantages in reliability, it has disadvantages in the number of round trips required to establish a secure connection. Before a browser sends a web page request, a secure connection will be needed. For creating an encrypted https connection, secure web browsing usually involves communicating over TCP, plus negotiating TLS. This approach needs at least two to three round trips with the server to establish a secure connection. With QUIC built on top of UDP, it is possible for a browser to start a connection and immediately start sending data without any acknowledgment from the remote side (0-RTT set up) as stated above in connection establishment section. Some other improvements on QUIC over TCP are the elimination of Head-of-line Blocking and Packet-Pacing.

a) Head-of-line Blocking: QUIC also eliminates head-of-line blocking. In TCP, head-of-line blocking can be seen because TCP checks the order in which packets are processed. It has to retransmit if the packet is lost on its way to the server. The TCP connection waits for the recovery packet before it can continue sending any other packets. In contrast, UDP doesn't check the order in which packets are received. QUIC exploits this property and layers a flexible stream multiplexing on top, in which only the contents of each stream is ordered[4]. When a packet of a stream is lost during transit, the entire connection is not blocked only one resource a file transmitted over that stream will be paused [4].

b) Packet-Pacing: TCP tries to send data as fast as possible. When data is sent too fast, losses are very likely to occur. Whenever loss occurs, we need to make retransition; this decreases the congestion window. Then the congestion window grows again until the next loss occurs. This causes bursty transmission. Packet lacing tries to make the transmission less bursty by not sending at full rate. This usually produces a positive effect on low bandwidth scenarios, but overall throughput for fast connections may decrease. TCP with some Linux kernels also uses packet pacing.

C. Security

TCP uses TLS for a secure connection. In addition to security benefits, TLS adds several additional round trips to each connection, making latency an even more crucial issue.

QUIC operates as an application protocol over User Datagram Protocol (UDP) and integrates ideas from Transmission Control Protocol (TCP), TLS, and Datagram Transport Layer Security (DTLS) to provide both security comparable to TLS and minimal latency during connection setup [5]. QUIC implements security by encrypting the application data and most of the protocol header. It provides improved latency with 0–RTT connection establishment [5]. Depending on whether TLS session resumption is enabled or not, the reduction in initial latency in comparison to TLS equals two or three round-trip times (RTTs). QUIC improves upon TCP and TLS in many other ways, too, including the introduction of multiple streams per connection to reduce head-of-line blocking stated above and vastly improved acknowledgment information that eliminates retransmission ambiguity issues Figure 3.

Some research work [5], presented a provable security analysis of QUIC that precisely characterizes its security guarantees. They proved that QUIC could successfully protect against such a strong attacker and provide QACCE security, under plausible assumptions about the encryption, authentication, and signing algorithms used[2].

D. Deployment

TCP is implemented in the OS kernel; users do not update their OS kernel very frequently and consequently. This restricts the frequency with which new changes or updates can be made to the TCP stack on an end-user's endpoint. QUIC is performed in the user-space, and for example in browsers, which update themselves every six weeks. Therefore deploying changes and updates to TCP can take multiple years.s. On the other hand, QUIC can evolve in weeks or months. Developers can unobstructedly test and experiment with new and innovative ideas.

E. QUIC Compared to SPDY

In this section, SPDY is discussed briefly. SPDY was developed by Google and improves the HTTP/1.1 Protocol. In Contrast to pure HTTP/1.1, SPDY allows hosts to send multiple HTTP/1.1 Requests/Responses within a single TCP segment. With SPDY we can prioritize HTTP/1.1 Requests/Responses. SPDY doesn't solve some of the problems because it is still using TCP. HTTP/2 is based on SPDY, and nowadays it is used more often than SPDY, SPDY has been deprecated in favor of HTTP/2. The performance of QUIC is evaluated by comparing the page load time of HTTP/1.1 over TCP, HTTP/1.1+ SPDY over TCP and HTTP/1.1 over QUIC.

V. Future of QUIC

QUIC's future seems to be tied in with demand for lower latency communications, and its adoption by the internet at large. Indeed in one of the early documents introducing QUIC, [8] states, "The number one goal of viability today is clearly a major driver for this protocol development." This desire for quick adaptability played a major role in the initial design and will also play a role in the future of QUIC. Large-scale adoption of QUIC seems dependent on its ability to provide a low latency but also a secure protocol. If QUIC can provide on its stated goals, there is potential for more widespread adoption. As mentioned, the current adoption is sizable at around 5-7% of all internet traffic, but a majority of that traffic is driven by Google's use of the protocol throughout their suite of applications and services [2]. The current demand for low latency and more responsive applications could prove positive to the wider adoption of QUIC. Additionally, a major design choice, to move much of the complexity out of the kernel layer, which was driven by the desire to be a viable option in a reasonable amount of time, makes the transition to QUIC simpler from a client perspective. In the future, as demand for rapid change and development increases, that decision also has the potential to positively benefit the wider adoption of the QUIC protocol.

It has recently been reported that major software companies other than Google and its parent company Alphabet have shown signs of adding support for the QUIC protocol [7]. One of the most recent categories of web applications is being called Progressive Web Apps. On Google's developer website they describe Progressive Web Apps as, "user experiences that have the reach of the web" The further specify that these Progressive Web Apps should give the feel of a native application while incorporating the advantages of traditional web-based applications [6]. If trends like Progressive Web apps continue, protocols, like QUIC, that offer the benefits of reduced latency and increased reliability could see an increase in the breadth of their deployment.

Ultimately [1]give the determination that, "Given the factors that impede QUIC support, the QUIC traffic share is likely to increase in the future when being largely enabled at a wide range of infrastructures." Which they earlier described as being made possible by the fact that, "QUIC paves the way towards a rapidly evolving transport that can be updated as easily and as frequently" [1] while still acknowledging the volatility and uncertainty of the future of the internet. In our experience we concur with [1], the QUIC protocol has laid the framework necessary for widespread adoption and has aligned itself well with the demands of the current internet market.

[1] J. Rüth, I. Poese, C. Dietzel, and O. Hohlfeld, “A First Look at QUIC in the Wild,” in *Lecture Notes in Computer Science*, 2018, pp. 255–268.

[2] A. Langley *et al.*, “The QUIC Transport Protocol,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication - SIGCOMM ’17*, 2017.

[3] L. Lumiaho, C. Liukas, L. Schulte, and J. Ott, “The Promise of QUIC: A Faster, More Flexible Transport Protocol — callstats.io.” [Online]. Available: https://www.callstats.io/2017/06/02/quic/. [Accessed: 13-May-2018].

[4] M. Thomson and J. Iyengar, “QUIC: A UDP-Based Multiplexed and Secure Transport,” Apr. 2018.

[5] M. Bishop, “Hypertext Transfer Protocol (HTTP) over QUIC,” Apr. 2018.

[6] “Progressive Web Apps | Web | Google Developers,” Google. [Online]. Available: https://developers.google.com/web/progressive-web-apps/. [Accessed: 14-May-2018].

[7] “Microsoft to add support for Google's QUIC fast internet protocol in Windows 10 Redstone 5,” Windows Latest, 02-Apr-2018. [Online]. Available: https://www.windowslatest.com/2018/04/03/microsoft-to-add-support-for-googles-quic-fast-internet-protocol-in-windows-10-redstone-5/. [Accessed: 14-May-2018].

[8] J. Roskind, "QUIC: Design Document and Specification Rationale," Google Slides. [Online]. Available: https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/mobilebasic. [Accessed: 14-May-2018].

References

Figure: 1

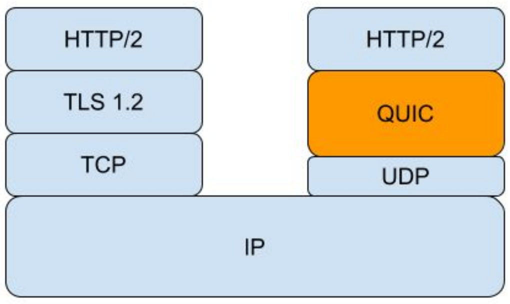


Figure: 2

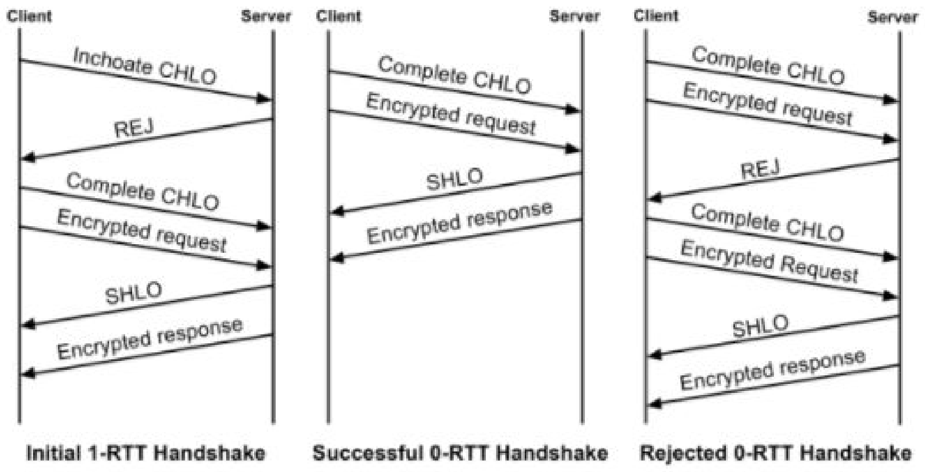
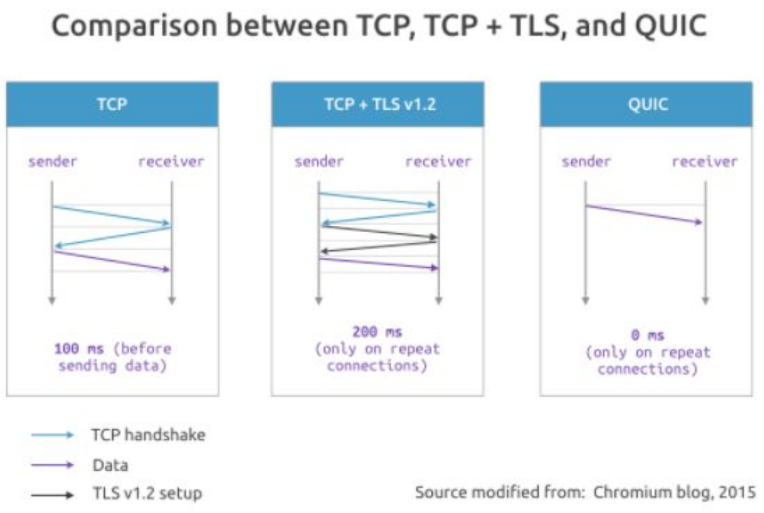


Figure: 3


Comparison between TCP, TCP + TLS, and QUIC
TCP
sender
receiver
100 ms (before sending data)
TCP + TLS v1.2
sender
receiver
200 ms (only on repeat connections)
QUIC
sender
receiver
0 ms (only on repeat connections)
TCP handshake
Data
TLS v1.2 setup
Source modified from: Chromium blog, 2015