Research on QUIC

April 13, 2023

(CS3072)

Submitted By:

Atish Kumar(120CS0173)

Shubham Dash(120CS0148)

Chandrajeet Manjhi(120CS0178)

Vedanta Hembram(120CS0159)

Ankur Kaushik(120CS0168)

Submitted To:

Dr. Sanjeev Patel

Table of Contents

The QUIC Transport Protocol: The Research write up.

ABSTRACT

We present our experience with QUIC a UDP Internet Connections is an experimental transport layer network protocol has been globally deployed at Google on thousands of servers and is used to serve traffic to a range of clients including a widely-used web browser (Chrome) and a popular mobile video streaming app (YouTube). We estimate that 7% of Internet traffic is now QUIC.

1. INTRODUCTION

QUIC is an encrypted, multiplexed, and low-latency transport protocol designed from the ground up to improve transport performance for HTTPS traffic and to enable rapid deployment and continued evolution of transport mechanisms.
QUIC replaces most of the traditional HTTPS stack: HTTP/2, TLS, and TCP (Figure 1).

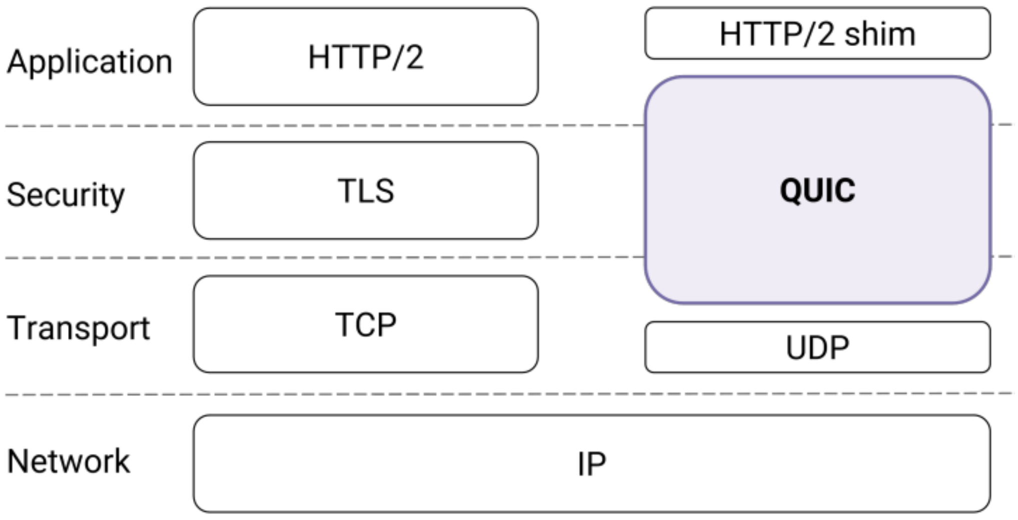


Figure 1: QUIC in the traditional HTTPS stack

We developed QUIC as a user-space transport with UDP as a substrate. QUIC eliminates head-of-line blocking delays by using a lightweight data-structuring abstraction, streams, which are multiplexed within a single connection so that loss of a single packet blocks only streams with data in that packet. On the server-side, our experience comes from deploying QUIC at Google’s front-end servers, which collectively handle billions of requests a day from web browsers and mobile apps across a wide range of services. On the client side, we have deployed QUIC in Chrome, in our mobile video streaming YouTube app, and in the Google Search app on Android. We find that on average, QUIC reduces latency of Google Search responses by 8.0% for desktop users and by 3.6% for mobile users, and reduces rebuffer rates of YouTube playbacks by 18.0% for desktop users and 15.3% for mobile users.

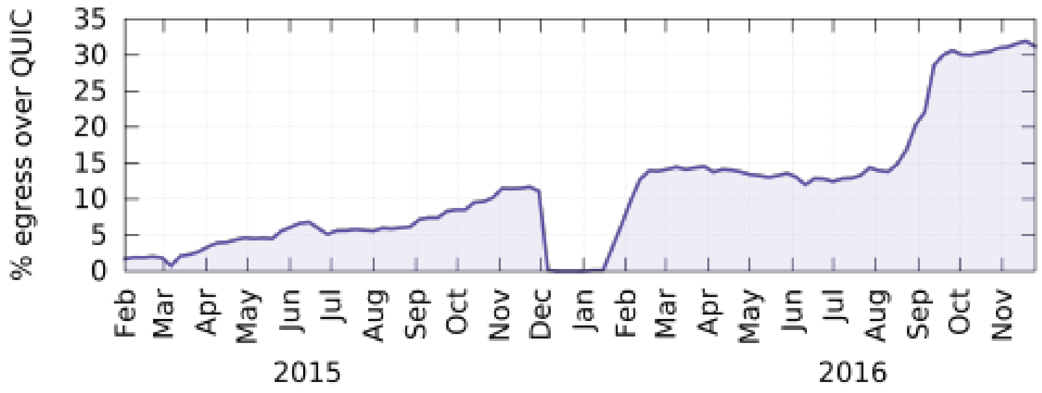


Figure 2: Timeline showing the percentage of Google traffic served over QUIC

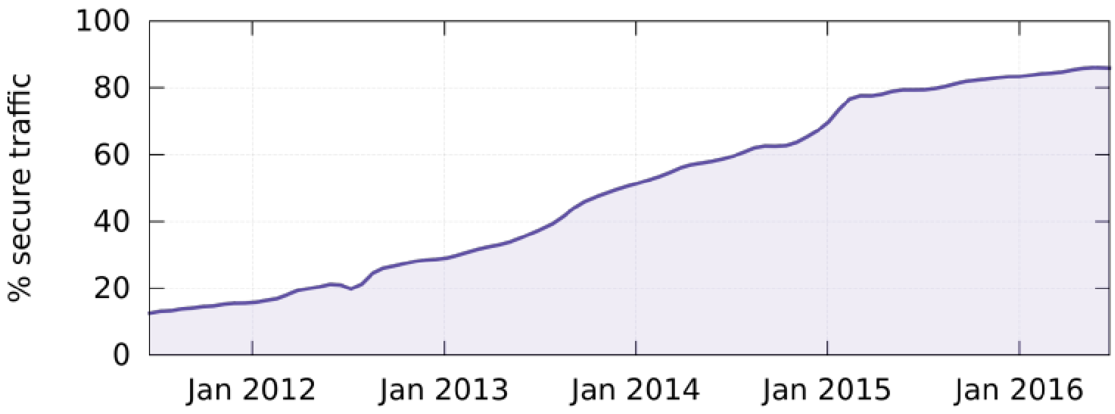


Figure 3: Increase in secure web traffic to Google's front-end servers.

2. MOTIVATION: WHY QUIC?

QUIC is designed to reduce latency in a few key ways, especially through improved multiplexing support, a decreased number of roundtrips, and improved packet loss handling. QUIC is implemented in the user space instead of the kernel space. The hope is that this will allow developers flexibility to improve congestion control over time, since it can be optimized and replaced and allows for rapid deployment. This differs from TCP, because TCP is implemented in the kernel space. This prevents flexibility, since kernel upgrades happen much less frequently than user upgrades. The hope is that if the application requires a specific implementation, the developer will now be able to provide it.

QUIC drastically reduces latency through a few methods.

2.1.1 Improved Multiplexing Support

This means that a single packet loss and retransmission packet causes head-of-line blocking (HOLB) for all the resources that are downloaded in parallel - the entire set of streams. The first packet holds up the rest of the line.

2.1.2. Fewer Round-trips

QUIC is focused on reducing the number of round-trips to establish a new connection. This includes the handshake step, encryption step, and initial data requests.

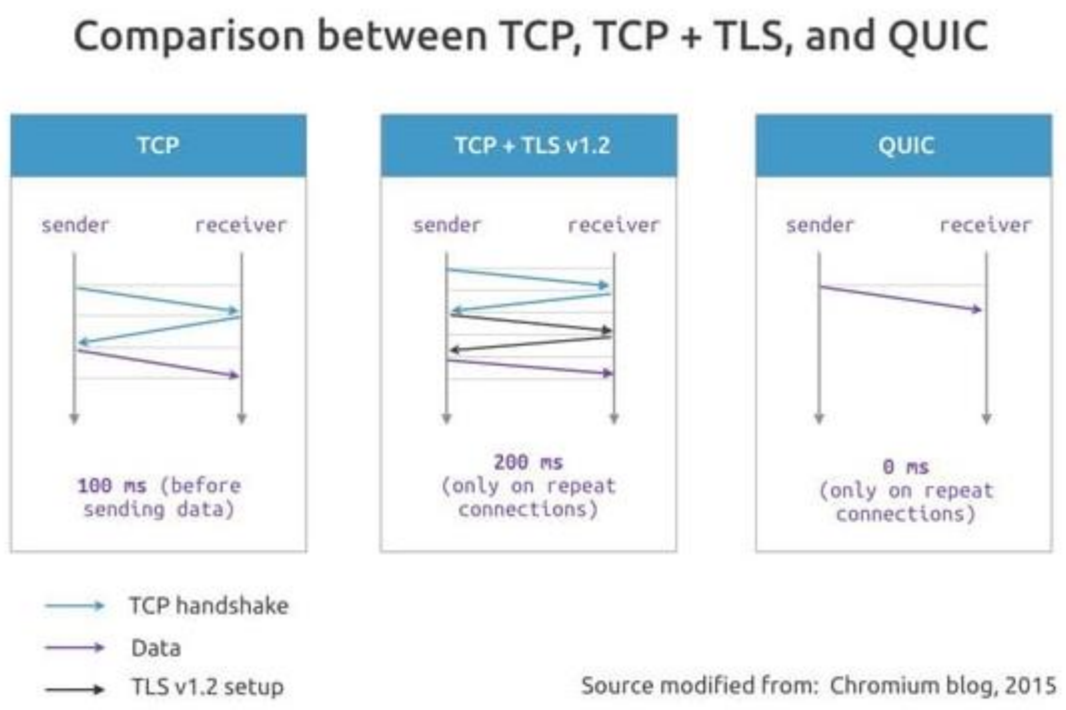

Comparison between TCP, TCP + TLS, and QUIC
TCP
sender
receiver
100 ms (before sending data)
TCP + TLS v1.2
sender
receiver
200 ms (only on repeat connections)
QUIC
sender
receiver
0 ms (only on repeat connections)
TCP handshake
Data
TLS v1.2 setup
Source modified from: Chromium blog, 2015

Figure show: TCP, TCP + TLS, & QUIC Trips

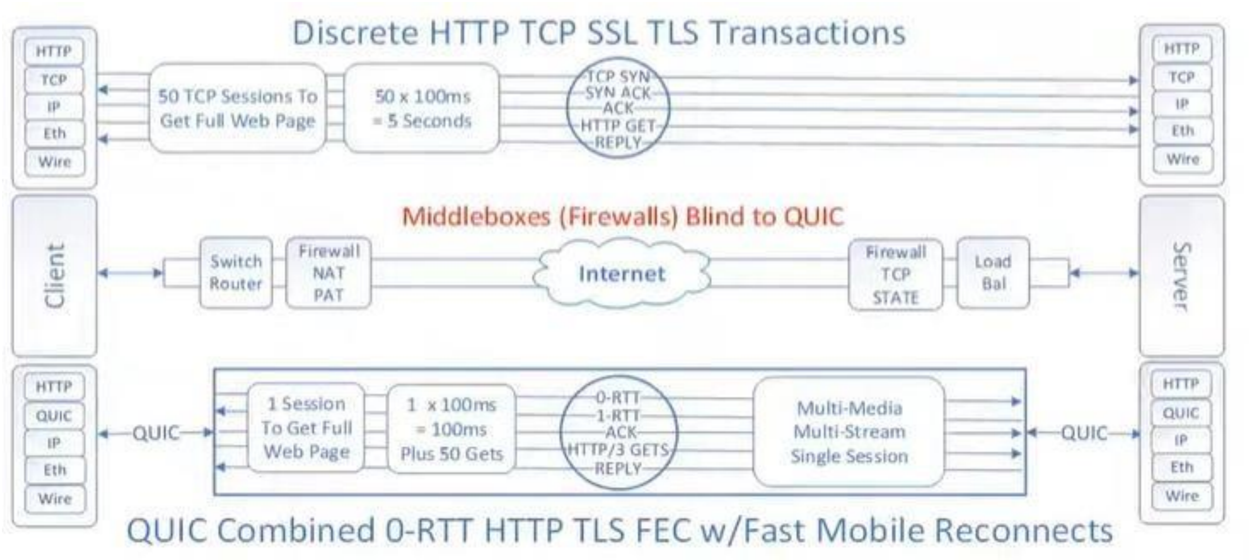
2.1.3. Improved Packet Loss Handling

QUIC is able to handle packet loss effectively due to several modern techniques.

- QUIC may align cryptographic block boundaries and packet boundaries to reduce packet loss.
- QUIC may use improved congestion control, including packet pacing based on ongoing bandwidth estimation.
- QUIC may send duplicates of the most critical packets, also known as proactive speculative retransmission.

3 .QUIC DESIGN AND IMPLEMENTATION

QUIC is designed to meet several goals, including deployability, security, and reduction in handshake and head-of-line blocking delays. It multiplexes multiple requests/responses over a single connection by providing each with its own stream, so that no response can be blocked by another. It improves loss recovery by using unique packet numbers to avoid retransmissionambiguity and by using explicit signaling in ACKs for accurate RTT measurements. It provides flow control to limit the amount of data buffered at a slow receiver and ensures that a single stream does not consume all the receiver's buffer by using per- stream flow control limits.


Discrete HTTP TCP SSL TLS Transactions
HTTP
TCP
IP
Eth
Wire
50 TCP Sessions To Get Full Web Page
50 x 100ms = 5 Seconds
TCP SYN
SYN ACK
ACK
HTTP GET
REPLY
Middleboxes (Firewalls) Blind to QUIC
Client
Switch Router
Firewall NAT PAT
Internet
Firewall TCP STATE
Load Bal
Server
QUIC
1 Session To Get Full Web Page
1 x 100ms = 100ms Plus 50 Gets
0-RTT
1-RTT
ACK
HTTP/3 GETS
REPLY
Multi-Media Multi-Stream Single Session
QUIC Combined 0-RTT HTTP TLS FEC w/Fast Mobile Reconnects

3.1 Connection Establishment

3.1.1 Initial Handshake: Initially, the client has no information about the server and so, before a handshake can be attempted, the client sends an inchoate client hello (CHLO) message to the server to elicit a reject (REJ) message.

The REJ message contains the following:

(i) a server config that includes the server's long-term Diffie-Hellman public value

(ii) a certificate chain authenticating the server

(iii) a signature of the server config using the private key from the leaf certificateof the chain

(iv) a source-address token: an authenticated-encryption block that contains the client's publicly visible IP address and a timestamp by the server.

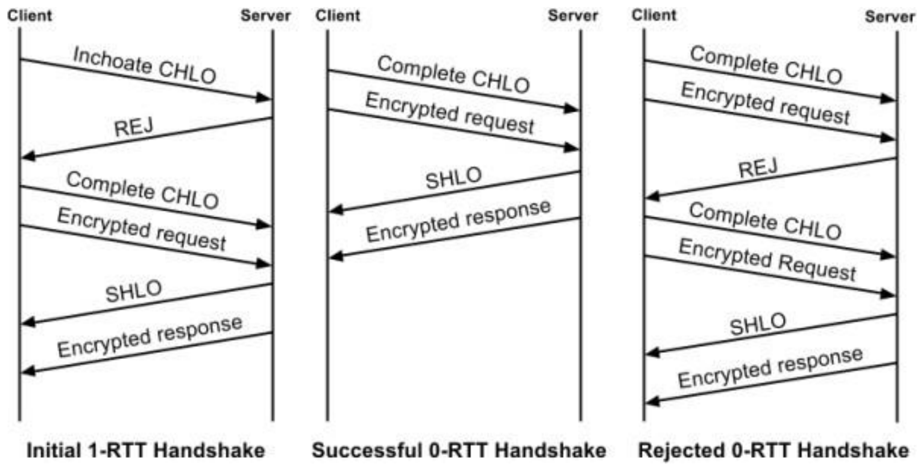
Once the client has received a server config, it authenticates the config by verifying the certificate chain and signature. It then sends a complete CHLO, containing the client's ephemeral Diffie-Hellman public value.

3.1.2 Final (and repeat) handshake: All keys for a connection are established using Diffie-Hellman. After sending a complete CHLO, the client is in possession of initial keys for the connection since it can calculate the shared value from the server's long-term Diffie Hellman public value and its own ephemeral Diffie-Hellman private key. At this point, the client is free to start sending application data to the server. Indeed, if it wishes to achieve 0-RTT latency for data, then it must start sending data encrypted with its initial keys before waiting for the server's reply.

If the handshake is successful, the server returns a server hello (SHLO) message. This message is encrypted using the initial keys, and contains the server'sephemeral Diffie -Hellman public value. With the peer's ephemeral public value in hand, both sides can calculate the final or forward-secure keys for the connection. Upon sending an SHLO message, the server immediately switches to sending packets encrypted with the forward-secure keys. Upon receiving the SHLO message, the client switches to sending packets encrypted with the forward-secure keys.

QUIC's cryptography therefore provides two levels of secrecy: initial client data is encrypted using initial keys, and subsequent client data and all server data are encrypted using forward-secure keys.

The client caches the server config and source-address token, and on a repeat connection to the same origin, uses them to start the connection with a complete CHLO. As hown in Figure 4, the client can now send initial-key-encrypted data to the server, without having to wait for a response from the server.



Eventually, the source address token or the server config may expire, or the server may change certificates, resulting in handshake failure, even if the client sends a complete CHLO. In this case, the server replies with a REJ message, just as if the server had received an inchoate CHLO and the handshake proceeds from there.

3.1.3 Version Negotiation: QUIC clients and servers perform version negotiation during connection establishment to avoid unnecessary delays. A QUIC client proposes a version to use for the connection inthe first packet of the connection and encodes the rest of the handshake using the proposed version. If the server does not speak the client-chosen version, it forces version negotiation by sending back a Version Negotiation packet to the client carrying all of the server's supported versions, causing a round trip of delay before connection establishment. This mechanism eliminates round-trip latency when the client's optimistically chosen version is spoken by the server, and incentivizes servers to not lag behind clients in deployment of newer versions. To prevent downgrade attacks, the initial version requested by the client and the list of versions supported by the server are both fed into the key-derivation function at both the client and the server while generating the final keys.

3.2 Stream Multiplexing:

Applications commonly multiplex units of data within TCP's single byte stream abstraction. To avoid head-of-line blocking due to TCP's sequential delivery, QUIC supports multiple streams within a connection, ensuring that a lost UDP packet only impacts those streams whosedata was carried in that packet. Subsequent data received on other streams can continue to be reassembled and delivered to the application.

QUIC streams are a lightweight abstraction that provide a reliable bidirectional bytestream. Streams can be used for framing application messages of arbitrary size—up to 264 bytes can be transferred on a single stream—but they are lightweight enough that when sending a series of small messages a new stream can reasonably be used for each one. Streams are identified by stream IDs, which are statically allocated as odd IDs for client-initiated streams and even IDs for server-initiated streams to avoid collisions. Stream creation is implicit when sending the first bytes on an as-yet unused stream, and stream closing is indicated to the peer by setting a "FIN" bit on the last stream frame. If either the sender or the receiver determines that the data on a stream is no longer needed, then the stream can be canceled without having to tear down the entire QUIC connection. Though streams are reliable abstractions, QUIC does not retransmit data for a stream that has been canceled.

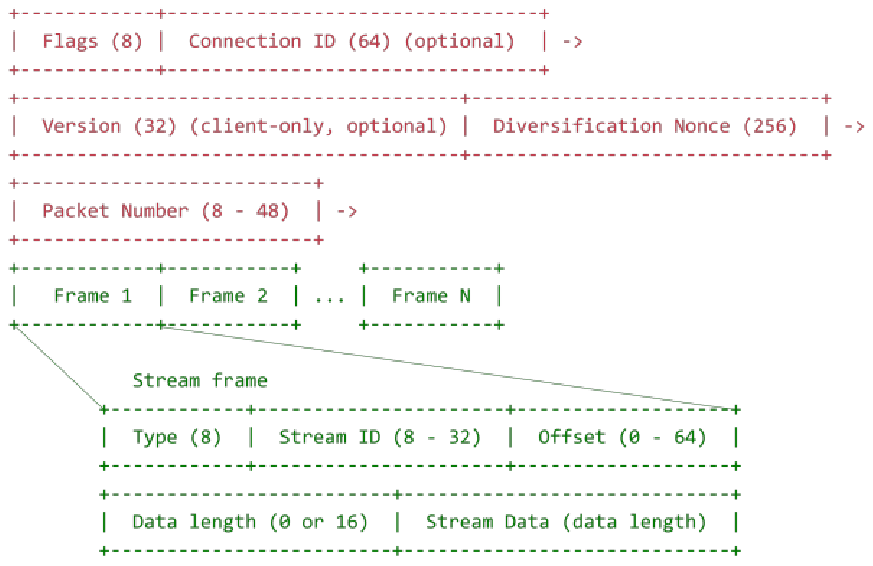


Figure 5: Structure of a QUIC packet, as of version 35 of Google's QUIC implementation. Red is the authenticated but unencrypted public header, green indicates the encrypted body. This packet structureis evolving as QUIC gets standardized at the IETF [2].

3.3 Authentication and Encryption:

With the exception of a few early handshake packets and reset packets, QUIC packets are fully authenticated and mostly encrypted. Figure 5 illustrates the structure of a QUIC packet. The parts of the QUIC packet header outside the cover of encryption are required either for routing or for decrypting the packet: Flags, Connection ID, Version Number, Diversification Nonce, and Packet Number4 . Flags encode the presence of the Connection ID field and length of the Packet Number field, and must be visible to read subsequent fields. The Connection ID serves routing and identification purposes; it is used by load balancers to direct the connection's traffic to the right server and by the server to locate connection state. The version number and diversification nonce fields are only present in early packets. The server generates the diversification nonce and sends it to theclient in the SHLO packet to add entropy into key generation. Both endpoints use the packet number as a per-packet nonce, which is necessary to authenticate and decrypt packets.

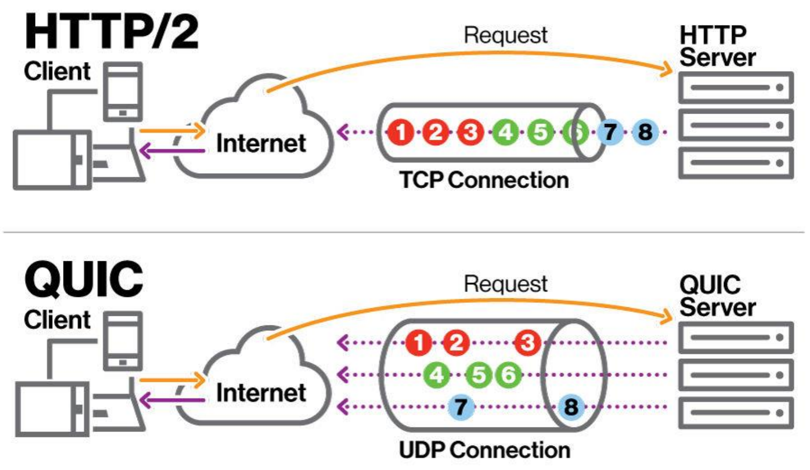
3.4 Loss Recovery

TCP sequence numbers facilitate reliability and represent the order in which bytes are to be delivered to the recipient. This merging causes a "re-transmission ambiguity" problem because the re-transmitted TCP segment carries the same sequence numbers as the original packet. A TCP ACK receiver cannot determine whether the ACK was sent for the original transmission or for re-transmission, and the loss of the re-transmitted segment is normally detected through an expensive timeout. Each QUIC packet carries a new packet number, including those carrying re-transmitted data. This design removes the need for a separate mechanism to distinguish a re-transmission ACK from the original transmission, thereby avoiding the TCP re-transmission ambiguity problem. Flow offsets within a flow are used for delivery ordering, separating the two functions that TCP brings together. The packet number represents an explicit time ordering that allows easier and more accurate detection of loss than in TCP
QUIC approval clearly indicates the delay between receiving the package and sending the ap proval. Combined with the uniformly increasing packet count, this provides an accurate network roundtrip time (RTT) estimation that aids in loss detection. An accurate RTT estimat ion can also help delay traffic management awareness such as BBR and PCC. Acknowledgem ent support up to 256 ACK blocks, making QUICmore resilient to preorders and drops than TCP with SACK.
Consequently, QUIC can keep more bytes on the wire in the presence of reordering or loss. These differences between QUIC and TCP allowed us to build simpler and more effective mechanisms for QUIC. We omit further mechanism details in this paper and direct the interested reader to the Internet-draft on QUIC loss detection.

3.5 Flow Control

Receivers should limit the amount of data they need to avoid poor delivery due to fast delivery inter fering with them or using too much memory. Streams are self-managed and Cross linked so that pools limit the memory Commitmentofconnections. The QUIC receiver controls the maximum amount of data a sender can send in a stream and passes all streams at once.The number of streams by limiting the number of buffers a stream can consume. Therefore,QUIC uses link-level flow control, which limits the bulk buffers a sender can use on receivers on all streams, and stream-level flow control, which limits the amount of buffers a sender can use in the stream. LikeHTTP/2, QUIC uses credit-based flow control.
The QUIC receiver reports the correct byte offset on each stream that the receiver is willing to recei ve data.

When data is sent, received, and transmitted in a stream, the receiver sends periodically update fra mes that increase the broadcast offset limit for Streams. peers are allowed to submit additional dat a. Link-level flow control works similarly to flow-level control, except that the number of bytes transmitted and the maximum offset received are summed across all streams.

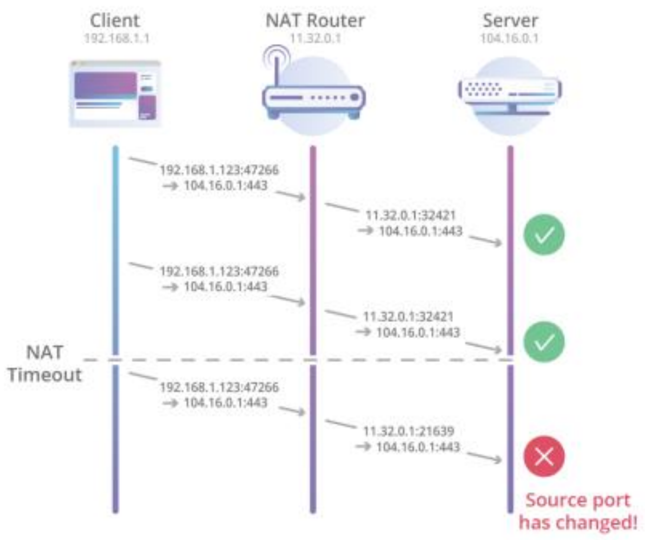


3.6 Congestion Control

The QUIC protocol is not tied to any control system and our application has a pluggable interface to allow testing. In our deployment, both TCP and QUIC use Cubic as a collision checker, with a notable difference.
For desktop and mobile streaming, our non-QUIC users use two TCP connections to the video server to receive video and audio data. These links are not designated as audio or video links;each audio and video block arbitrarilyuse one of the two connections.Because audio and video streams are sent over two streams in a QUIC connection, QUIC uses the difference between multT CP and Cubic in the congestion avoidance phase, which uses TCP for parity checking for stream integrity.

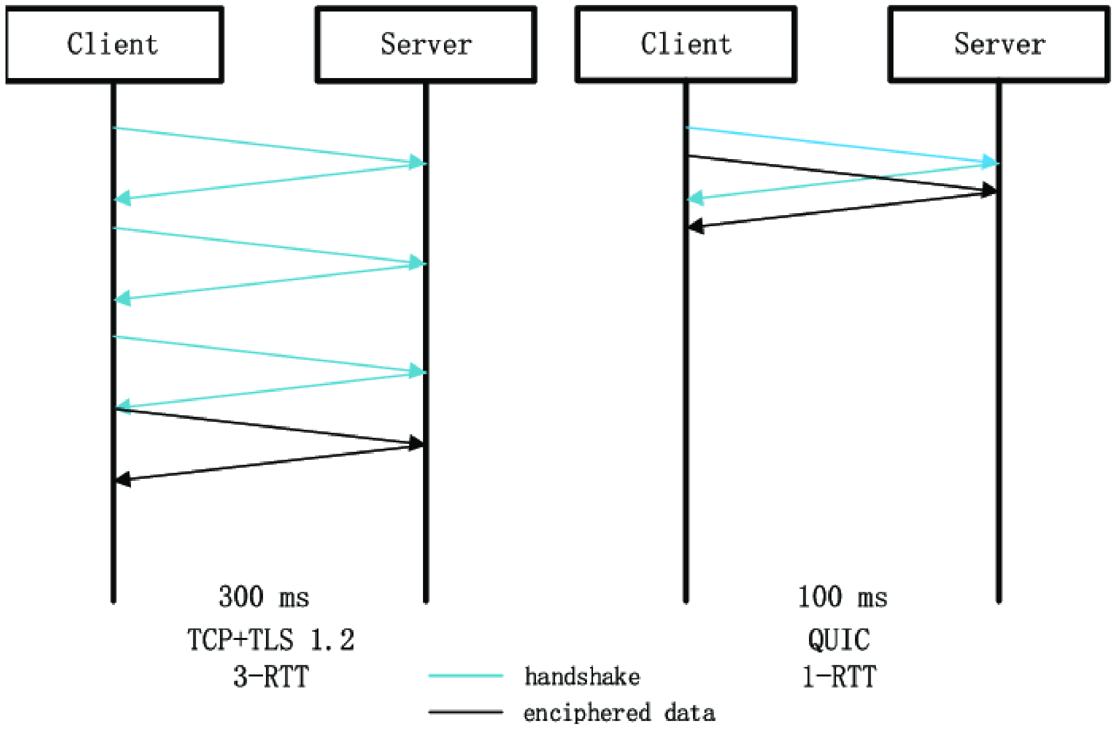
3.7 NAT Rebinding and Connection Migration

QUIC connections are identified by a 64-bit connection ID.
QUIC's connection ID ensures that connections survive client IP and port changes.
These changes canbe caused by NAT timeouts and redirection (more common for UDP than TCP) or by clients changing network connections to new IP addresses. When the QUIC endpoint simply eliminates the NAT duplicationproblem by using the connection ID to identify the connection,the us er initiates the relay connection. is now an event with limited distribution.



3.8 QUIC Discovery for HTTPS

The client does not know in advance whether the server is using QUIC.When our client makes the first HTTP request to the root,sends the request over TLS/TCP. Our servers announce QUIC support by adding the "Alt-Svc" header to their HTTP response .This header tells the client that it can try to connect to the resource using QUIC. The user can now try using QUIC at after referencing the same origin.
In subsequent HTTP requests to the same resource, client competes with QUIC and TLS/TCP connections, but prefers QUIC,connections, delaying the TLS/TCP connection by about 300 milliseconds.Which protocol completed the connection before ending the request. If QUIC is blocked on the route or if the QUIC handshake packet is larger than the route MTU, then the QUIC cannot be processed and the client connects with reverse TLS/TCP.



4. QUIC PERFORMANCE

In this section, we define three key application metrics that drove QUIC's development and deployment, and we describe QUIC's impact on these metrics. Desktop version and mobile version differ in operating environment as well as a difference in implementation. The Google Search and YouTube apps were developed independently from Chrome, and while they share the same networkstack implementation, they are tuned specifically for the mobile environment. For example, the Google Search app retrieves smaller responses, whose content has been tailored to reduce latencyin mobile networks. Similarly, the YouTube app pre-warms connections to reduce video playback latency and uses an Adaptive Bit Rate (ABR) algorithm that is optimized for mobile screens.

Tables 1 and 2 summarize the difference between QUIC users and TLS/TCP users on three metrics: Search Latency, Video Playback Latency, and Video Rebuffer Rate. For each metric, the tables show QUIC's performance impact as a percent reduction between using TLS/TCP and using QUIC. If QUIC decreased Search Latency from 100 seconds to 99 seconds, it would be indicated as a 1% reduction.

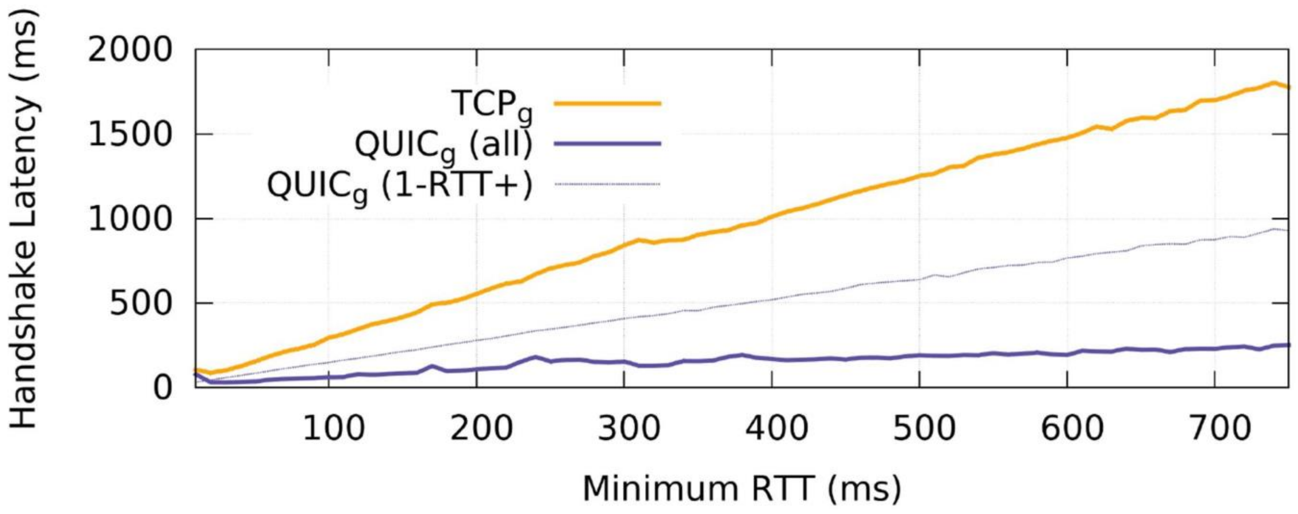
4.1 Experiment Setup

Users are either in the QUIC experimental group (QUICg) or in the TLS/TCP control group (TCPg)..The size of the QUICg and TCPg populations are equal throughout.

Clients that do not use QUIC use HTTP/2 over a single TLS/TCP connection for Search and HTTP/1.1 over two TLS/TCP connections for video playbacks. Both QUIC and TCP implementations use a pacedform of the Cubic algorithm for congestion avoidance. We show data for desktop and mobile users, with desktop users accessing services through Chrome, and mobile users through dedicated appswith QUIC support. Unless otherwise noted, all results were gathered using QUIC
version 35 and include over a billion samples.

4.2Transport and Application Metrics

Handshake latency is the amount of time taken to establish a secure transport connection. In TLS/TCP, this includes the time for both the TCP and TLS handshakes to complete. We measured handshake latency at the server as the time from receiving the first TCP SYN or QUIC client hello packet to the point at which the handshake is considered complete. In the case of a QUIC 0-RTT handshake, latency is measured as 0 ms. This figure shows the impact of QUIC's 0-RTT and 1-RTThandshakes on handshake latency.



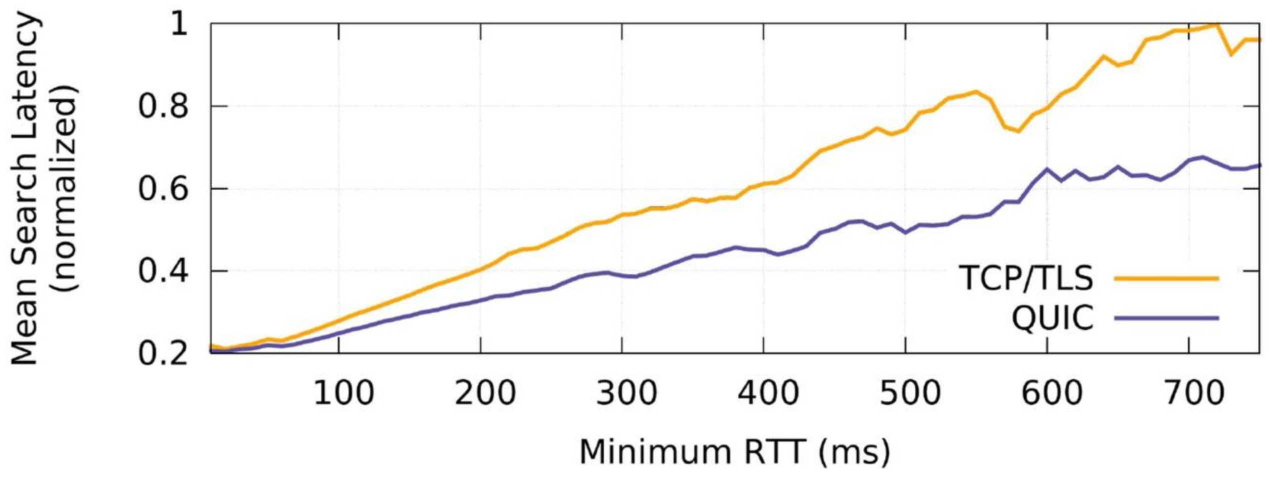
Comparison of handshake latency for QUICg and TCPg versus the minimum RTT of the connection. Solid lines indicate the mean handshake latency for all connections, including 0-RTT connections. The dashed line shows the handshake latency for only those QUICg connections that did not achievea 0-RTT handshake.

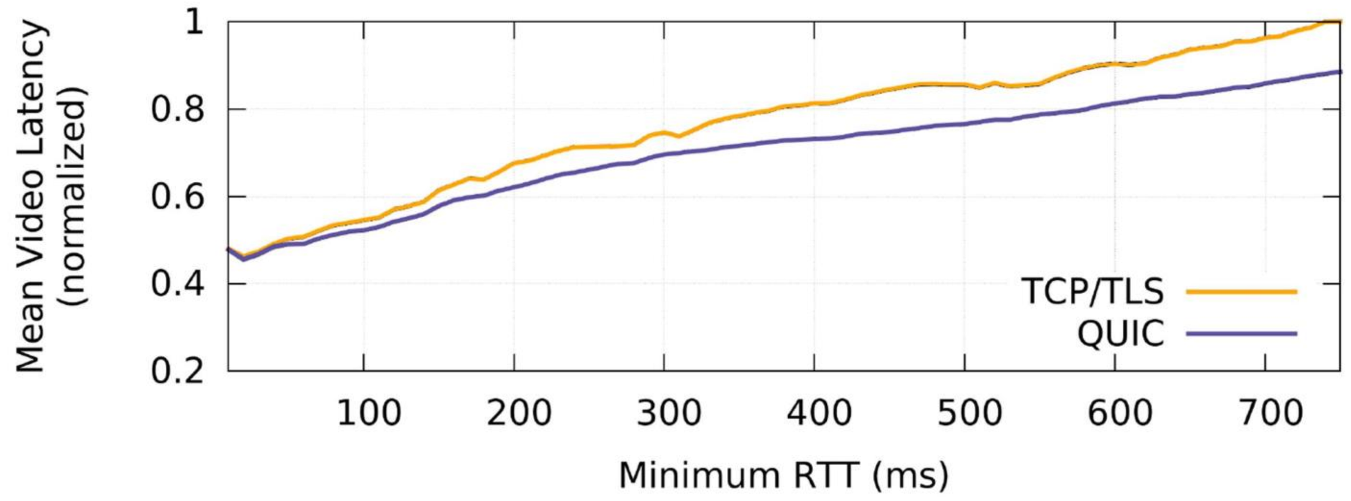
With increasing RTT, average handshake latency for TCP/TLS trends upwards linearly, while QUIC stays almost flat. QUIC's handshake latency is largely insensitive to RTT due to the fixed (zero) latency cost of 0-RTT handshakes, which constitute about 88% of all QUIC handshakes. The slight increase in QUIC handshake latency with RTT is due to the remaining connections that do not successfully connect in 0-RTT.

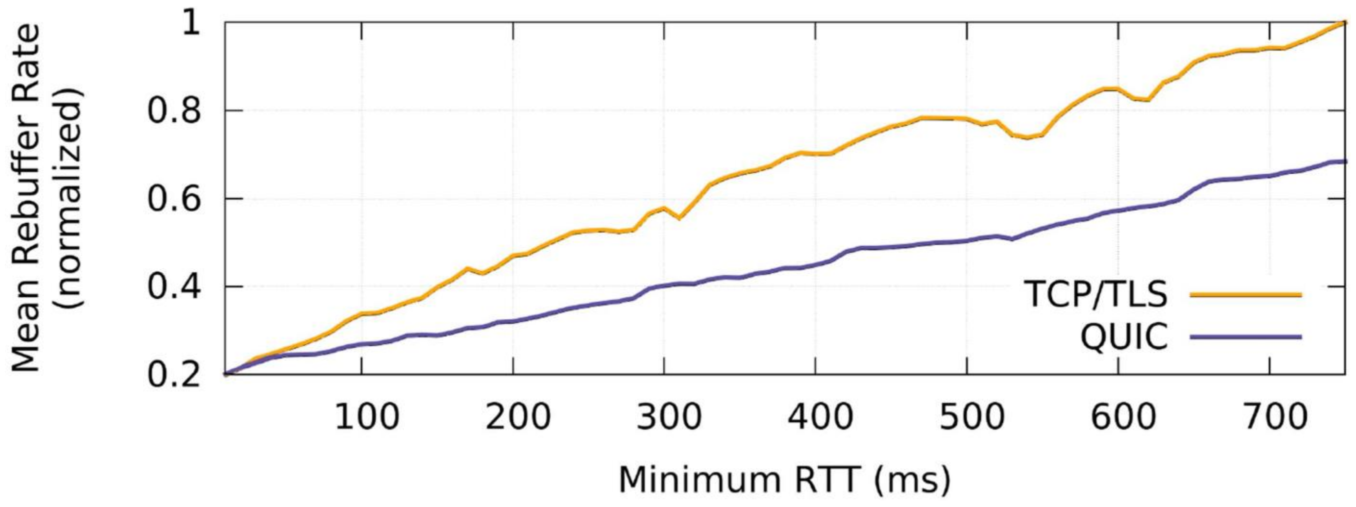
4.3 Search Latency

Search Latency is the delay between when a user enters a search term and when all the search- result content is generated and delivered to the client by Google Search, including all correspondingimages and embedded content. On average, an individual search performed by a user result in a total response load of 100 KB for desktop searches and 40 KB for mobile searches. As a metric, Search Latency represents delivery latency for small, delay-sensitive, dynamically-generated payloads.

As shown in Table 1, users in QUICg experienced reduced mean Search Latency. The percentile data shows that QUICg's improvements increase as base Search Latency increases. This improvement comes primarily from reducing handshake latency. As the user's RTT increases, the impact of savinghandshake round trips is higher, leading to larger gains in QUIC.

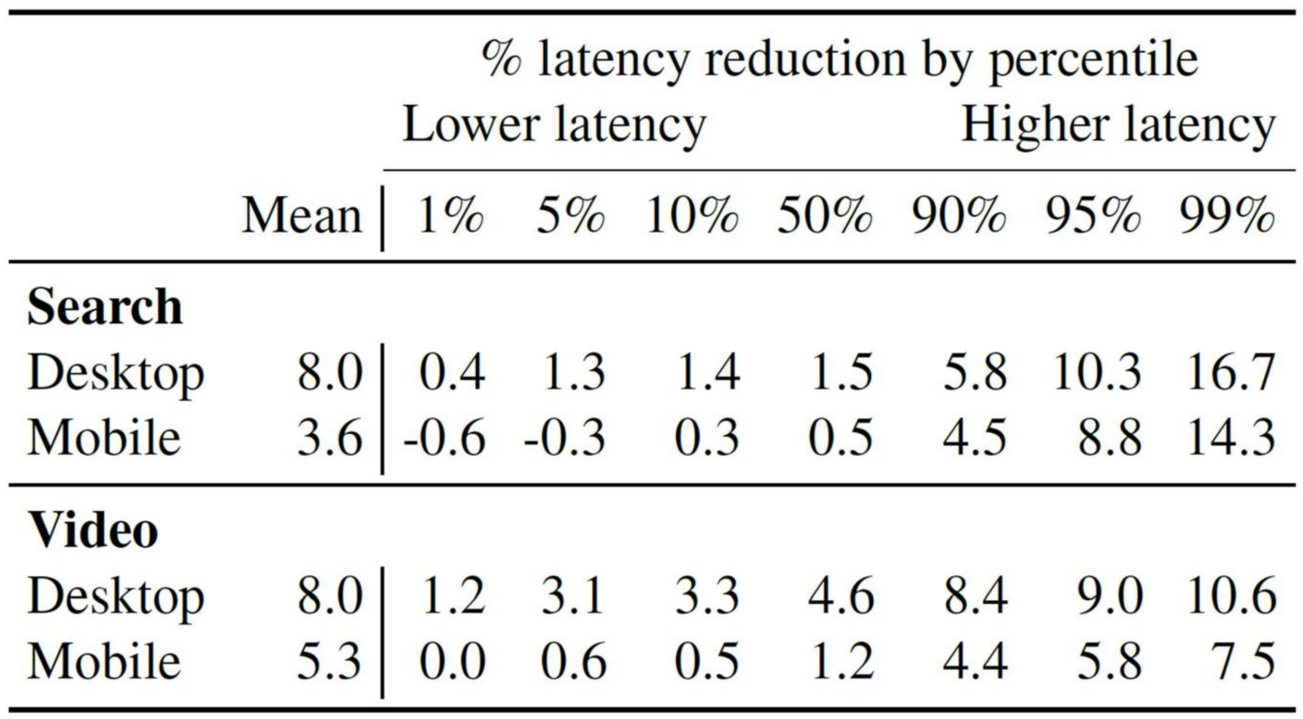






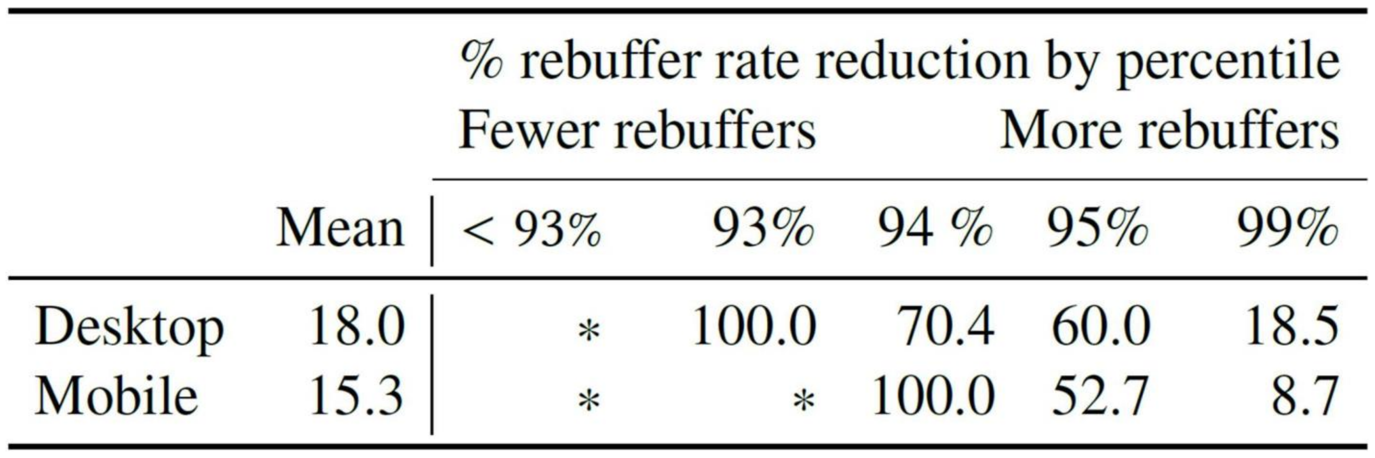
Comparison of QUICg and TCPg for various metrics, versus minimum RTT of the connection. The y-axis is normalized against the maximum value in each dataset. Presented data is for desktop, but the same trends hold for mobile as well. The x-axis shows minimum RTTs up to 750 ms.

Figure below further shows that users with high RTTs are in a significant tail: more than 20% of all connections have a minimum RTT larger than 150ms, and 10% of all connections have a minimum RTTlarger than 300ms. Of the handshake improvements, most of the latency reduction comes from the0-RTT handshake: about 88% of QUIC connections from desktop achieve a 0-RTT handshake, which is at least a 2-RTT latency saving over TLS/TCP. The remaining QUIC connections still benefit from a 1-RTT handshake.



	% latency reduction by percentile							
		Lower latency						Higher latency
	Mean	1%	5%	10%	50%	90%	95%	99%
Search								
Desktop	8.0	0.4	1.3	1.4	1.5	5.8	10.3	16.7
Mobile	3.6	-0.6	-0.3	0.3	0.5	4.5	8.8	14.3
Video								
Desktop	8.0	1.2	3.1	3.3	4.6	8.4	9.0	10.6
Mobile	5.3	0.0	0.6	0.5	1.2	4.4	5.8	7.5

Table 1: Percent reduction in global Search and Video Latency for users in QUICg, at the mean and atspecific percentiles. A 16.7% reduction at the 99th percentile indicates that the 99th percentile latency for QUICg is 16.7% lower than the 99th percentile latency for TCPg.

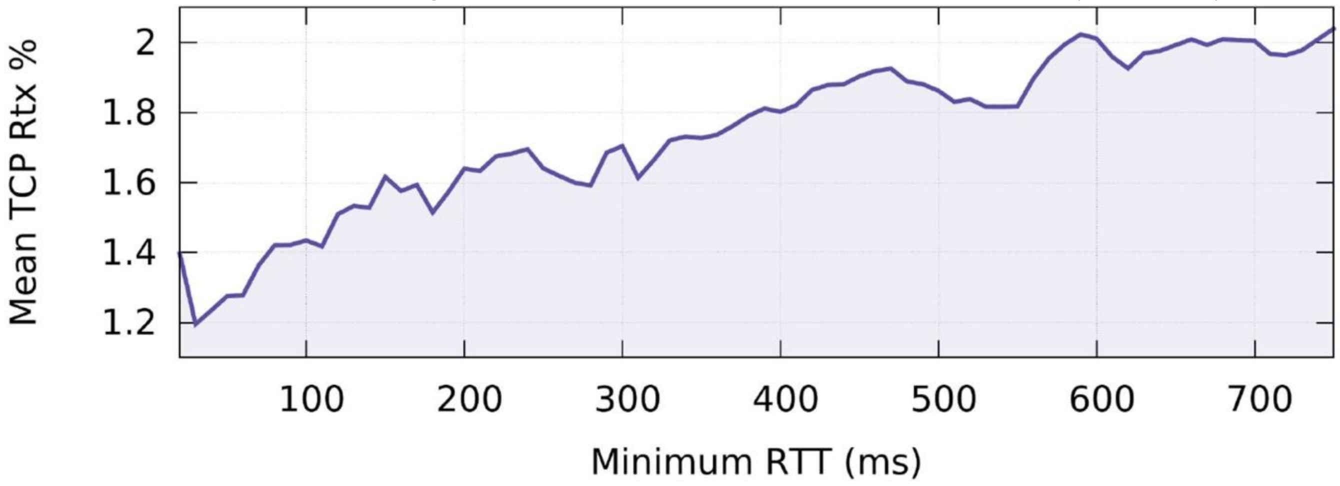


	% rebuffer rate reduction by percentile					
		Fewer rebuffers				More rebuffers
	Mean	< 93%	93%	94 %	95%	99%
Desktop	18.0	$*$	100.0	70.4	60.0	18.5
Mobile	15.3	$*$	$*$	100.0	52.7	8.7

Table 2: Percent reduction in global Video Rebuffer Rate for users in QUICg at the mean and atspecific percentiles. An 18.5% reduction at the 99th percentile indicates that the 99th percentile rebuffer rate for QUICg is 18.5% lower than the 99th percentile rate for TCPg. An $*$ indicates thatneither QUICg nor TCPg have rebuffers at that percentile.

Table 1 shows that Search Latency gains on mobile are lower than gains on desktop. In addition to differences between desktop and mobile environments and usage, the lower gains are explained in part by the fact that QUIC connections established by the mobile app only achieve a 68% 0-RTT handshake rate on average—a 20% reduction in successful 0-RTT handshake rate as compared to desktop. This can be due to two factors. First, when mobile users switch networks, their IP addresschanges, which invalidates the source-address token cached at the client. Second, different server configurations and keys are served and used across different data centers. When mobile users switch networks, they may hit a different data center where the servers have a different server config than that cached at the client.

We believe that QUIC's loss recovery mechanisms may also play a role in decreasing Search latencyat higher RTTs. Recall that QUIC includes richer signaling than TCP, which enables QUIC loss recovery to be more resilient to higher loss rates than TCP. Figure below shows the relationship between TCP retransmission rates measured against minimum client RTTs. This figure shows that network quality is highly correlated with the client's minimum RTT. Consequently, QUIC's improvedloss recovery may also contribute to Search Latency improvement at higher RTTs
Average TCP retransmission rate versus minimum RTT observed by the connection. this graph shows the retransmission rate averaged within 10 ms RTT buckets, and the actual rate experiencedby a



connection can be much higher. Across all RTTs, retransmission rates are 0%, 2%, 8% and 18% at the 50th, 80th, 90th and 95th percentiles.

4.4 Video Latency

Video latency for a video is measuredas the time between when a user hits'play' on a video to the video starts playing. The amount of data the Player loads depends on the bitrate ofthe playback. It is observed that whenuser is in QUIC, It experienced a overall video latency.
Video Latency gains increase with theclient RTT.

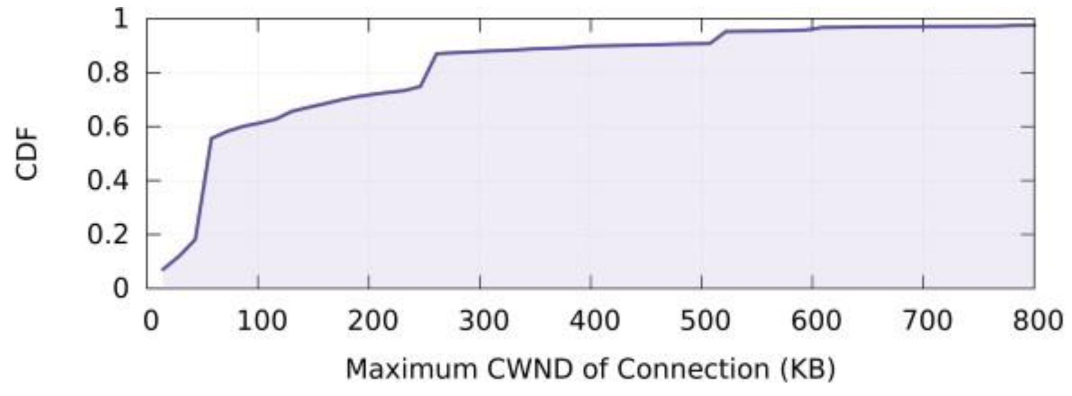
QUIC loss recovery improvements may helpvideo latency as client RTT increases.

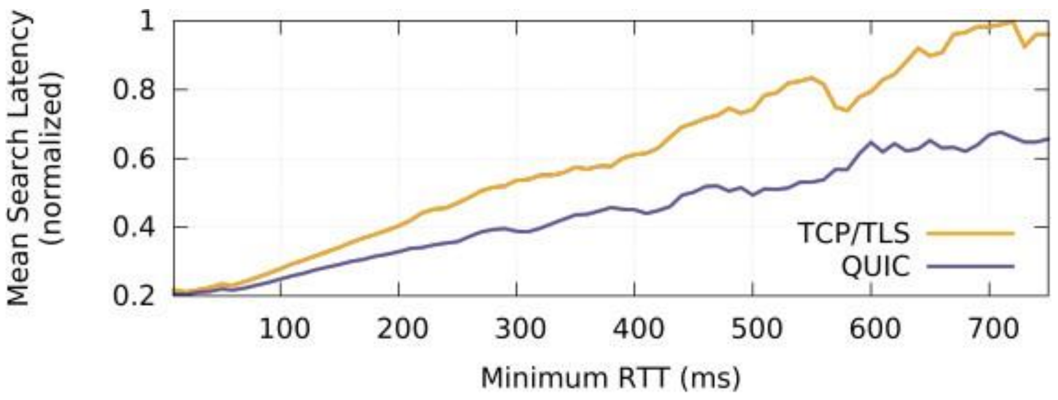
QUIC benefits mobile playbacks less than the d2esktop.

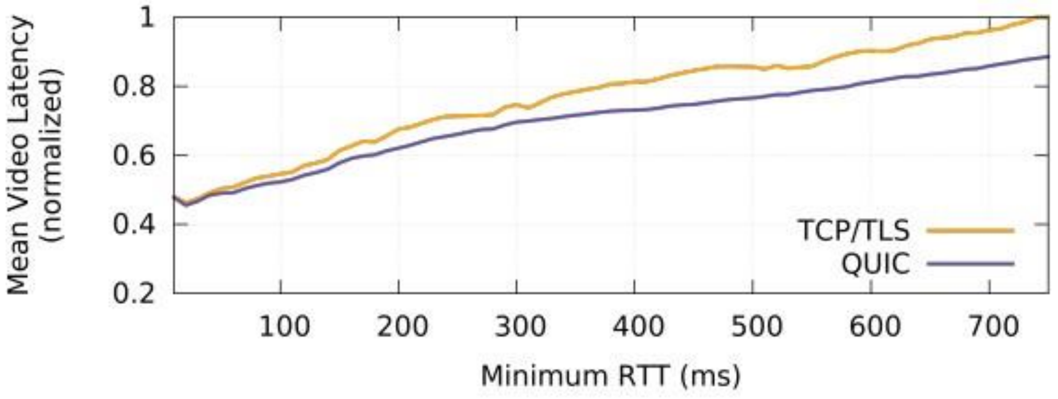
4.5 Video Rebuffer rate

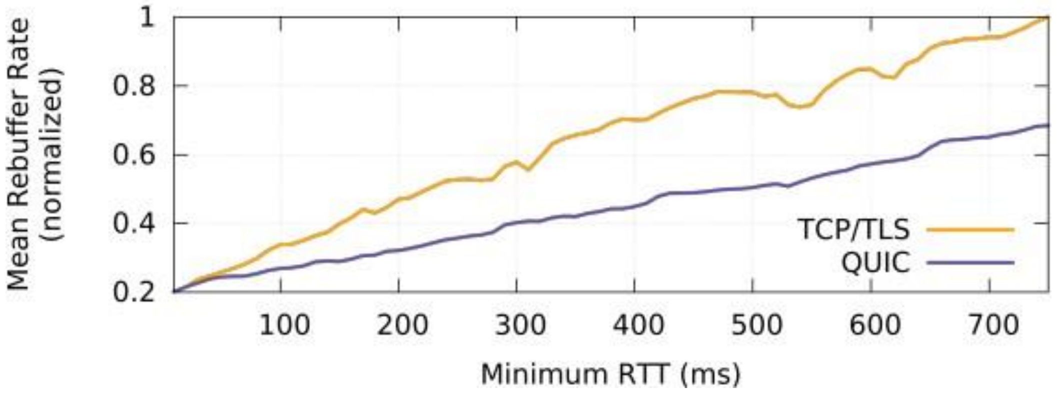
Video Rebuffer rate is the percentage of time that a video pauses during playback ,the video pauses during a playback can rebuffer data .

- Users in QUIC experienced reducedrebuffer rate on average reductions
- Both QUIC and TCP rebuffer rate increaseswith higher RTT .
- The default initial connection –level flow control limit advertised by the QUIC clientis 15MB .
- QUIC increased the number of videos played at the optimal rate by 2.9% for desktops and 4.6% for moblile playbacks .


CDF
Maximum CWND of Connection (KB)
0
0.2
0.4
0.6
0.8
1
0
100
200
300
400
500
600
700
800

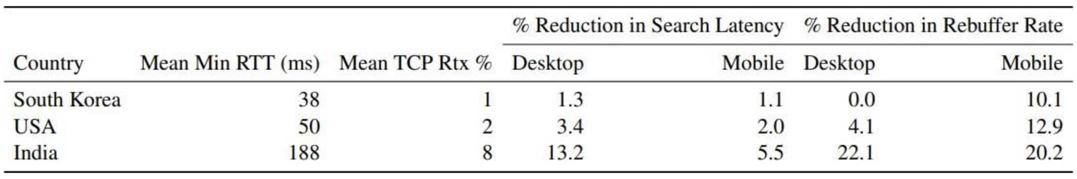

Mean Search Latency (normalized)
Minimum RTT (ms)
TCP/TLS
QUIC
0.2
0.4
0.6
0.8
1
100
200
300
400
500
600
700


Mean Video Latency (normalized)
Minimum RTT (ms)
TCP/TLS
QUIC
0.2
0.4
0.6
0.8
1
100
200
300
400
500
600
700


Mean Rebuffer Rate (normalized)
Minimum RTT (ms)
TCP/TLS
QUIC
0.2
0.4
0.6
0.8
1
100
200
300
400
500
600
700

4.6 Performance By Region

We can now see QUIC performance impacton various countries . This impact is differentfor different countries and regions .



			% Reduction in Search Latency		% Reduction in Rebuffer Rate	
Country	Mean Min RTT (ms)	Mean TCP Rtx %	Desktop	Mobile	Desktop	Mobile
South Korea	38	1	1.3	1.1	0.0	10.1
USA	50	2	3.4	2.0	4.1	12.9
India	188	8	13.2	5.5	22.1	20.2

From the table we can see that QUIC performance is closer to that of TCP network condition in United States .

QUIC shows greater improvements in USAthan in South Korea .

Benefits are higher in the regions that havehigher RTT.

4.7 Server CPU Utilization

It was found that QUIC server utilization wasabout 3.5 times higher than TLS/TCP .

Three major sources of QUIC CPU cost are :

- Cryptography
- Sending and Receiving UDP Packets
- Maintaining internal QUIC state

With these optimizations, there is decreasein CPU cost of serving web traffic over QUIC to approximately twice that of TLS/TCP