Game of protocols: Is QUIC ready for prime time streaming?

Article *in* International Journal of Network Management · February 2019

DOI: 10.1002/nem.2063

3 authors:

Şevket Arısu
Ozyegin University
2 PUBLICATIONS **4** CITATIONS
SEE PROFILE

Ertan Yildiz
Özyeğin University
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Ali C. Begen
Ozyegin University
99 PUBLICATIONS **2,558** CITATIONS
SEE PROFILE

Received 15 July 2018; | Revised 25 October 2018; | Accepted 10 January 2019

DOI: xxx/xxxx

PREPRINT VERSION

Game of Protocols: Is QUIC Ready for Prime Time Streaming?

Sevket Arisu[1] | Ertan Yildiz[1] | Ali C. Begen[2]

[1]Ozyegin University and Turkcell, Turkey
[2]Ozyegin University, Turkey

Correspondence
Sevket Arisu Email: sevket.arisu@ozu.edu.tr

Present Address
This is the pre-peer reviewed version of the following article: [1], which has been published in final form at [https://doi.org/10.1002/nem.2063]. This article may be used for non-commercial purposes in accordance with Wiley Terms and Conditions for Use of Self-Archived Versions.

Summary

QUIC (Quick UDP Internet Connections) is an experimental and low-latency transport protocol proposed by Google, which is still being improved and specified in the IETF. The viewer's quality of experience (QoE) in HTTP adaptive streaming (HAS) applications may be improved with the help of QUIC's low-latency, improved congestion control and multiplexing features. We measured the streaming performance of QUIC on wireless and cellular networks in order to understand whether the problems that occur when running HTTP over TCP can be reduced by using HTTP over QUIC. The performance of QUIC was tested in the presence of network interface changes caused by the mobility of the viewer. We observed that QUIC resulted in quicker start of media streams, better streaming and seeking experience, especially during the higher levels of congestion in the network and had a better performance than TCP when the viewer was mobile and switched between the wireless networks. Furthermore, we measured QUIC's performance in an emulated network that had a various amount of losses and delays to evaluate how QUIC's multiplexing feature would be beneficial for HAS applications. We compared the performance of HAS applications using multiplexing video streams with HTTP/1.1 over multiple TCP connections to HTTP/2 over one TCP connection and to QUIC over one UDP connection. To that effect, we observed that QUIC provided better performance than TCP on a network that had large delays. However, QUIC did not provide a significant improvement when the loss rate was large. Finally, we analyzed the performance of the congestion control mechanisms implemented by QUIC and TCP, and tested their ability to provide fairness among streaming clients. We found that QUIC always provided fairness among QUIC flows, but was not always fair to TCP.

KEYWORDS:
HTTP adaptive streaming, DASH, QUIC, QoE, HAS.

1 | INTRODUCTION

Today, the QUIC protocol is distributed and extensively being used in many Google services. Over 30% of Google's total egress traffic is QUIC[2]. QUIC is widely deployed to the user space by the Chrome browser (both desktop and mobile), and already comprises 7% of all Internet traffic[3]. QUIC aims to reduce the connection establishment latency and improve the congestion control, and provides multiplexing streams and encryption at application transport level. QUIC replaces most of the traditional

HTTPS stack: HTTP/2, TLS and TCP. The architecture of QUIC is shown in Figure 1 . Some of these features may be modified or removed, or new features may be added over time as all the aspects regarding the QUIC protocol are currently being specified in the IETF by a large number of experts across a number of companies and researchers working in the field.

The common belief that HTTP must be used over TCP is a misconception. Although TCP has been the primary protocol to use with HTTP, the HTTP specifications (RFCs 7230 and 7540) do not state that using TCP is mandatory. Rather, HTTP must be used with a reliable transport protocol. When TCP is used, if a packet loss occurs, TCP holds the subsequent packets that may have already arrived in the client's buffer. In other words, TCP does not deliver the already received packets to the application layer unless the missing packet is recovered through a retransmission in order to guarantee in-order delivery.

With respect to HTTP adaptive streaming, head-of-line blocking and slow retransmissions could sometimes cause the delays of delivering media segments and degrade viewer experience, in particular if the data in the playback buffer of the player is running out. QUIC can solve this problem. Multiplexed streams are used over one UDP connection to support multiple HTTP requests in parallel by QUIC. Streams in QUIC are independent of each other and each stream provides a reliable delivery. In the case of losing a packet in a stream, the other streams are not affected. HTTP/2 (RFC 7540) has multiplexing features, too. However, HTTP/2 may still have the problem of head-of-line blocking if it is used over TCP. Another feature of QUIC is the reduced handshake latency. For the subsequent connections, TCP requires a three-way handshake and thus, requires 1.5 round-trip times (RTT) before any data request is received at the server. QUIC, on the other hand, requires 0.5 RTT before a data request is received at the server. QUIC uses a combined cryptographic and transport handshake for setting up a connection. On a successful handshake, a client saves information about the origin (host, port and URI scheme) in its cache. In the subsequent connections to the same origin, the client can establish a connection with no additional round trips and data can be sent immediately following the client handshake without waiting for a reply from the server. Thus, QUIC connection establishment takes 0 RTT when a server is known by the client (see Figure 2). This feature may potentially reduce the initial startup or seeking latency for HAS applications. Last but not least, QUIC may help to get a higher throughput by the help of its improved loss recovery and congestion control features.

Previous studies show that viewers are quite sensitive to rebuffering events as even one percent increase in rebuffer rate can trigger a decrease in watch time of more than three minutes[4]. Another research[5] showed that the viewer engagement linearly decreased with increasing rate of rebuffer rate up to a certain threshold. Viewers watched only 30% of the video when there were more than 0.3 buffering events per minute. Experiencing more buffering events, the viewers stopped watching the rest of the content. Buffering events can take place when there is a significant delay or packet loss in the network during the playback, and also when the viewer wants to seek to a video frame that has not been downloaded yet. Buffering events can also occur when the network interface changes (*e.g.*, switching from WiFi to LTE or 3G network). During this change, the IP address of the device changes and the streaming client has to re-establish the connection.

In our previous work[6], we examined how QUIC and TCP performed during connection changes due to viewer's mobility and also we focused on the seeking performance of these two protocols in the open Internet. In this study, we rather examine how QUIC's multiplexing feature can be beneficial for HAS applications in an emulated network. Additionally, we measure how fair TCP and QUIC clients are when they are competing with other clients. We try to answer the following questions: First, how does QUIC's multiplexing feature perform against HTTP/1.1 over multiple connections and HTTP/2 over a single connection when there are random losses and delays in the network? Second, how fair TCP and QUIC clients are when they are competing with other clients while the network has various amounts of losses and delays.

This study is structured as follows: In Section 2, we present an overview of other relevant studies that looked into streaming over QUIC. Section 3 describes our approach and setup that is designed to compare the performance of streaming clients running QUIC and TCP under different frame-seek and connection-switch scenarios as well as our approach for evaluating QUIC's multiplexing feature in a controlled network. In Section 3, we also investigate the fairness aspects. In Section 4, we include an analysis of the observed QoE measurements. Finally, Section 5 presents our conclusions, and a discussion of potential benefits of QUIC transport for HAS along with the future work.

2 | RELATED WORK

Timmerer *et al.* discovered that using QUIC instead of TCP, was not effective for the overall streaming performance considering increased or decreased media throughput[7]. Conversely, Szabo *et al.* proved QUIC's benefit in initial buffering time with a range of 6-49% changing on the media properties and network environment[8]. Li *et al.* worked on an MMT (MPEG Multimedia

Transport) based multimedia system using QUIC and found that QUIC was a better option for media transport in comparison with HTTP when using on an MMT system[9]. Bhat *et al.* compared TCP versus QUIC at the transport layer and measured the performance of the HAS adaptation algorithms. The authors discovered that QUIC was not beneficial for the existing adaptation algorithms because they were designed with respect to TCP[10]. Zinner *et al.* also worked on the same issue and found that QUIC with 0-RTT connection establishment clearly performed better than the other protocols, reducing the start time for the playback[11]. According to Google's reports, QUIC reduced the rebuffer rates of YouTube by 18% and 15% for desktop users and mobile users, respectively[2]. Kakhki *et al.* reported that QUIC provided better streaming QoE than TCP, but only for high-definition video[12]. Ayad *et al.* worked on the performance of commercial and open-source players and discovered that the QUIC protocol was quite aggressive when competing with other TCP flows and not as responsive to congestion as other TCP flows[13].

Some studies examined QUIC for not necessarily media streaming over HTTP but ordinary web transport. Carlucci *et al.* discovered QUIC had a better performance than TCP, considering web page load times when there were no random losses and QUIC performed better than SPDY (the pre-standard version of HTTP/2) when there were losses in the network[14]. Magyesi *et al.* investigated QUIC's web performance over HTTP/1.1 and SPDY, and found that they were not exactly better than each other and the network conditions determined which protocol gave better performance[15]. Cook *et al.* founded out that QUIC outperformed HTTP/2 over TCP/TLS in wireless mobile networks[16]. Qian *et al.* worked on web content delivery on mobile networks and the authors found that multiple QUIC connections could cope with the restrictions of different congestion control algorithms when downloading short-lived web content[17].

Viernickel *et al.* worked on MPQUIC, a multipath extension for the emerging QUIC transport protocol. They found that MPQUIC reduced download times and with multipathing support, QUIC was ready to become the universal stream transport protocol in today's Internet[18]. In their study, Coninck *et al.* found that without packet losses, while the performance of single-path TCP and single-path QUIC were similar, multipath-QUIC could outperform multipath-TCP[19]. Yang *et al.* worked on QUIC's performance in space-terrestrial integrated networks compared with TCP and TCP-ECN (improved through Explicit Congestion Notification). The authors found that QUIC outperformed in all scenarios with the help of its zero handshaking and loss recovery strategies, while QUIC behaved poorly where there was a short-term disconnection during handover[20]. Hussein *et al.* studied an SDN (Software Defined Network) for QUIC and the authors found that the SDN improved resource utilization and network security when integrated with QUIC[21].

According to the result of some research, QUIC was not as competitive as TCP in a network with little loss, large buffer or large propagation delay[22]. Furthermore, there was no markable increase for adaptive streaming performance with QUIC[10,7]. We suspect that these low-performance results of QUIC are primarily due to the fact that these studies tested either an open-source (experimental) implementation[23],[24] that was not provided by Google or an older version of Google's QUIC implementation. The QUIC's specification is being rapidly developed and deployed to the user space. For that reason, a non-Google code or an older version of the code is not guaranteed to reflect the true performance of QUIC. Google informs that their QUIC toy server and toy client are not "performant at scale"[25]. However, in our evaluations, we saw that the performance was good enough for testing with one or two clients.

Interested readers may refer to[6] to see the detailed comparison of the previous studies as well as the specific software used in those studies.

3 | APPROACH AND ENVIRONMENT SETUP

We made a set of modifications in the Python-based player[26] to make it able to work with QUIC protocol. We tested frame-seek and connection-switch scenarios on the public Internet (uncontrolled environment). On the other hand, we used a testbed (controlled environment) to evaluate different multiplexing techniques and analyze the performance of QUIC's congestion control mechanism as well as its ability to provide fairness.

3.1 | Player Modifications and Used Algorithms

We made some modifications in the Python-based player[26] to ensure a fair comparison of QUIC and TCP. First, we integrated Google's QUIC client into the player as a sub-process due to the lack of a Python library implementation for QUIC. The media segments were downloaded by this sub-process. Second, the original player[26] uses Python's *urllib*[27] to make HTTP requests and it creates a new connection for each media segment. However, QUIC transfers the data over a single UDP connection by default.

To avoid openning multiple connections when streaming over TCP, a new TCP client was integrated into the player instead of using Python's *urllib*. The TCP client was built using *libcurl*[28] and worked with HTTP keep-alive feature on. We also enabled Apache HTTP server's *KeepAlive* feature. Figure 4 shows how the HAS player works with TCP and QUIC. The Python player communicates with the clients via a sub-process[29] and it requests the related media file from the clients. Upon a request from the player, the TCP or the QUIC client communicates with the operating system via C++ libraries, downloads the requested file and sends the size information of the downloaded file to the player. Finally, we made some changes in Google's QUIC server code to disable its in-memory cache in order to ensure a fair comparison. The modified player code, modified QUIC server code, the QUIC and TCP clients are available for public access on GitHub for the research community[30].

In our study, we used the following adaptation algorithms to run the experiments:

- **BASIC (Throughput based):** This adaptation algorithm uses the average of the segment download rates. It starts by requesting the segment with the lowest bitrate and then it selects the bitrate for the next segment based on the calculated average throughput[26].
- **SARA - Segment Aware (Buffer based):** SARA considers the segment size variation in addition to the estimated bandwidth and the current buffer occupancy to accurately determine the time required to download the next segment. This algorithm predicts the throughput with a weighted harmonic mean method[26].
- **BBA-2 (Buffer based):** BBA-2 algorithm uses a set of functions that maps the buffer occupancy to a bitrate. The algorithm tries to reduce the rebuffer events and increase the average bitrate quality. It directly chooses the next bitrate considering the buffer occupancy and only uses bandwidth estimation if necessary. This algorithm was used in a wide-scale Netflix experiment[31].

Note that in our study we primarily focused on the transport options for HTTP adaptive streaming, not the features of the particular bitrate adaptation algorithms.

3.2 | Internet Setup

For uncontrolled experiments, we did not use any test-bed or any traffic shaping tool to throttle the network speed, produce loss or delay. Instead, we ran the experiments on the Internet. We used the QUIC server (v39) provided by Google[25] for QUIC and Apache's HTTP server (v2.4.33)[32] for TCP. We set up the servers on an Amazon EC2 instance in Frankfurt (Germany) and the streaming clients were in Istanbul (Turkey). The clients were connected via WiFi to residential broadband access network or via a smartphone tethered to a commercial LTE or 3G network.

Table 1 and Figure 3 show the network characteristics observed during the tests. The lower, middle and upper bars in the box plots in Figure 3 represent the 25^{th}, 50^{th} and 75^{th} percentiles of the measured RTTs, respectively, for all three types of wireless networks. For WiFi, LTE and 3G networks, the average RTTs (drawn as black squares) are 69 ms, 128 ms and 234 ms, respectively. Note that the RTTs that were above 400 ms for 3G network are not shown in Figure 3 .

For the uncontrolled environment experiments, we ran the tests for three different adaptation algorithms on three types of wireless networks (WiFi, LTE and 3G) in different time slots. However, to ensure that both protocols (QUIC and TCP) faced the same conditions in the Internet, we started the players for each protocol with a time offset of half a second.

3.2.1 | Frame-Seek Scenario

This section describes our approach to evaluate the performance of HAS applications over QUIC and TCP for several frame-seek events. In our study, the results and conclusions that we present are not only applicable for forward frame-seek scenarios, but also for backward frame-seek scenarios.

We implemented frame-seek ability in the player to measure the performance of QUIC and TCP when the viewer wanted to seek to a frame in the video. Upon jumping to a second in the content where the respective media segments have not been downloaded yet, there will not be any segment in the playback buffer and the player will require to fill it up quickly in order to have a smooth seeking experience.

Table 2 shows the frame-seek scenario that has four seeking events at different seconds. The player jumps to the seek-to time when the playback time equals to seek-at time. Generally, the adaptation algorithm of the player may download the segments at a lower bitrate to shorten the seeking time at the cost of decreased playback quality. In our evaluations, we set the player to keep

the bitrate equal to the one of the last played segment before the seeking request since our primary goal was to observe the impact of the transport layer on the seeking experience. We measured the seconds between the time the frame seek was requested and the playback time of the requested segment. An on-demand video was used in these frame-seek scenario evaluations.

3.2.2 | Connection-Switch Scenario

We installed a script at the client machine to evaluate QUIC's performance against various network interface disconnections and reconnections. Within fixed intervals, the client's active network interface was changed from WiFi to LTE or 3G or vice versa by this script. The player tried to reconnect to the Internet by using the new interface when it detected a connection loss. The playback continued as long as there was at least one media segment in the playback buffer during the connection re-establishment process. The connection re-establishment process may degrade the QoE, increase rebuffering rate and produce a stall, naturally. In our tests, we measured the rebuffer rate and the average playback bitrate when the client has various reconnection events that were caused by network interface changes.

Most connections can be re-established in a few seconds thanks to the improvements in the networking stack. The player can cope with most reconnection events without resulting in a rebuffering event or significant quality degradation when it is streaming an on-demand content with a large playback buffer. The player may have to decide to download representations that are encoded at lower bitrates if the reconnection process takes a long time. However, if the reconnection takes a longer time to complete, the playback will inevitably stall. We used a live video content and a smaller buffer size for the player for better understanding the impact of using QUIC vs. TCP during the network interface switches. Figure 5 shows the network setup for these evaluations and the connection switch scenario is shown in Table 3 .

3.3 | Test-Bed Setup

In this section, we focus on evaluating QUIC's multiplexing feature and also its ability to provide fairness in a controlled environment. For the controlled experiments, our setup consisted of two machines, one for the players and one for the servers (HTTP and QUIC). The network topology shown in Figure 6 consists of a server and a client connected through a bottleneck link of 10 Mbps. The empirical RTT from the client to the server is 1 ms and the packet loss rate is negligible. The server and client are bare metal machines that run vanilla Ubuntu 16.04. We emulated the network in the bottleneck link between the router and server shown in Figure 6 using the Network Emulator (tc-NetEm) tool[33]. We used the emulation tool to throttle the available bandwidth of the link between the client and the server according to the throughput profiles. We used FCC[34] and DASH-IF NP2b[35] throughput profiles to throttle the bandwidth in the emulated tests. The FCC profile[34] provides a profile with a bandwidth variation of 0.8 Mbps to 1.4 Mbps, while the DASH-IF NP2b profile[35] provides a bandwidth variation of 1.5 Mbps to 5 Mbps. The average bandwidth of FCC and DASH-IF NP2b profiles is 1.1 Mbps and 3 Mbps, respectively. Furthermore, we also emulated the respective RTT and packet loss rate. We applied four different delay and loss patterns (Typical Delay - Typical Loss), (Typical Delay - Large Loss), (Large Delay - Typical Loss) and (Large Delay - Large Loss) to stress the TCP and UDP stack as recommended in the DASH-IF guidelines[35]. The emulated network conditions were applied to the link between the client and server within fixed intervals (30 seconds), while the client has been playing the video. The parameters of network profiles were taken from the DASH-IF guidelines and N2b profile is shown in Table 4 . We used an on-demand content from a dataset[36] in these tests. The segment duration and playback buffer size were four and 20 seconds, respectively.

3.3.1 | Evaluating QUIC's Multiplexing Feature

One technique used to send multiple streams of information is stream multiplexing. This is done over a single transport connection. Since HTTP/1.1 can request just one resource at a time, web browsers generally open multiple independent TCP connections at the same time. For example, Google Chrome opens up to six connections and Internet Explorer (v11.0) opens up to 13 connections per origin at the same time to handle multiple requests to each domain[37]. However, this produces additional delays and increases the complexity of the application. HTTP/2 has multiplexing features to alleviate head-of-line blocking, but may still suffer from it, if HTTP/2 is used over a single TCP. In HTTP/2, head-of-line blocking may be caused by missing data on one stream for the data successfully transmitted and received on another stream. This is because, despite the independence of each payload data of the different streams, they are still transmitted in order, to the application over the same TCP connection. QUIC has multiplexing features as well. It uses multiplexed streams on one UDP connection to handle concurrent requests in

parallel. Each stream in QUIC has reliable delivery, independent of each other, which are also delivered in order. The streams are not affected if a packet gets lost in another stream.

We evaluated the following three different approaches of sending multiple streams of data for HAS applications as shown in Figure 7 :

(a) HTTP/1.1 over a varying number of TCP connections,

(b) HTTP/2 with parallel requests over a single TCP connection, and

(c) QUIC over a single UDP connection.

While using HTTP/1.1 over multiple TCP connections, each video segment was downloaded by more than one TCP connection using the HTTP partial GET method. We experimented with different numbers of TCP connections such as one, two, four, six, eight, ten and twelve connections to download a single segment. When using more than one connection, each segment file was divided into two, four, six, eight, ten or twelve equal parts, and each part was downloaded by a separate TCP connection at the expense of increased overhead. The specification for HTTP/1.1 recommends limiting the number of connections opened by a client to any server (although no specific limit is expressed)[38]. In the HTTP/2 approach, we used *libcurl*[28] HTTP/2 implementation with multiplexing and pipe-lining features enabled. The partial bytes of a media segment were downloaded using a single TCP connection by sending multiple HTTP/2 GET requests in parallel without waiting for the response from the previous request. QUIC can multiplex streams as well. It multiplexes multiple requests and responses over a single UDP connection by providing each with its own stream, so that no response can be blocked by another. In its QUIC implementation, Google chose 1350 bytes as the default payload size for QUIC. Based on Google's experiments, there is a rapid decrease in reachability after 1450 bytes, which is caused by the total packet size (sum of QUIC payload, UDP and IP headers) exceeding the 1500 byte Ethernet maximum transmission unit (MTU) limit[2]. In our experiments, we did not change this setting. The maximum packet size for QUIC was 1392 bytes (including the headers) in our tests.

In these experiments, multiplexing features were used to download a single segment, not multiple segments at the same time. This way, our approach did not *steal* throughput from future segments.

3.3.2 | Evaluating QUIC's Fairness

One of the most important elements to provide fair and high utilization of Internet links shared by multiple clients is transport-layer congestion control. The essential feature of transport-layer protocols is that they do not use more than their fair share of bandwidth. An unfair protocol may lead performance degradation for competing clients. In this section, we evaluate fairness aspects when multiple streaming clients compete for network resources in the controlled environment. This experiment is designed to analyze the performance of the congestion control mechanisms implemented by TCP and QUIC, and test their ability to provide fairness to all streaming clients. The CUBIC congestion control[39] algorithm is used in both TCP and QUIC implementations. QUIC uses a variant of mulTCP for CUBIC during the congestion avoidance phase[2].

We evaluated the following scenarios and we applied four different network emulation patterns for the ten-minute video. We repeated the experiments ten times and we present the averages in Figures 8 , 10 , 9 and 11 .

(a) One TCP client, not competing with any client,

(b) One QUIC client, not competing with any client,

(c) Two TCP clients, competing with each other,

(d) Two QUIC clients, competing with each other,

(e) One TCP and one QUIC client, competing with each other.

4 | RESULTS

In our experiments, we used a 600-second long content encoded with AVC codec at a constant bitrate[36]. The bitrates ranged from 44 Kbps to 3.9 Mbps in the media presentation file. For the on-demand content, the segment duration and playback buffer size were four and 20 seconds, respectively. For the live content, the segment duration and playback buffer size were set to two and eight seconds, respectively. In our evaluations, each test was repeated 10 times to prevent substantial anomalies. In this section, we present the average results.

4.1 | Metrics

We measured following metrics in our tests:

- *Average Playback Bitrate*: We measured the average of the bitrates of the segments downloaded during the tests. Generally speaking, higher encoding bitrate implies better QoE. We note that the average playback bitrate must be considered together with other metrics such as the number of bitrate changes to make a more accurate assessment of the QoE[40].
- *Average Wait Time after Seeking*: This is the time from the frame-seek request to the playback of the requested media. A rule of thumb is to keep this time under two seconds[41,42]. We measured this metric only for the frame-seek scenarios.
- *Rebuffer Rate*: The rebuffer rate is calculated as follows:

$$\text{Rebuffer Rate} = \frac{\text{Rebuffer Time}}{\text{Rebuffer Time} + \text{Media Play Time}} \quad (1)$$

  where the Rebuffer Time is the time that the media pauses during the playback to rebuffer and Media Play Time is the length of the media.

4.2 | Results for the Frame-Seek Scenario

The frame-seek results for the BASIC, SARA and BBA-2 adaptation algorithms are shown Tables 5 , 6 and 7 , respectively. QUIC achieved a shorter average wait time after seeking and started the media streams more quickly. QUIC also reduced the wait times by up to 50% compared to TCP. QUIC reduced the rebuffer rates as well. Our results may seem to conflict with some of the previous research[10]. We suspect that its contradictory nature (compared to ours) is primarily due to the fact that an open-source server implementation with experimental QUIC support[23,24] was used by Bhat *et al.*

The player may have to empty the whole playback buffer upon a frame-seek request. The BASIC algorithm does not consider the current buffer occupancy to calculate the next bitrate. However, the SARA and BBA-2 algorithms are sensitive to buffer drains. During the tests, the available bandwidth on all types of wireless networks was sufficient to download the segments at the highest bitrate. For these reasons, the BASIC algorithm gives a better average playback bitrate than the other two algorithms. The SARA and BBA-2 algorithms may be modified to better cope with the frame-seek events.

For all algorithms on all types of wireless networks, we saw that the average wait time after seeking was almost two seconds or longer when streaming over TCP. When streaming over QUIC, average wait times were reduced to less than one and half a second. This is an important result, considering that keeping the average seek time less than two seconds is essential for viewer engagement and loyalty[41,42].

We observed that the rebuffer rates were reduced by QUIC as well. When streaming over TCP, the rebuffer rates were higher than 3% for the WiFi and LTE networks, and higher than 5% for the 3G network. When streaming over QUIC, the rebuffer rates dropped to 1% and 2% for the WiFi and LTE networks, respectively. QUIC reduced the rebuffer rate more dramatically from 6% to 2% for the 3G network, which naturally has delays larger than the other networks. In our tests, the 3G network has 82% higher average RTT than the LTE network, and 339% higher average RTT than the WiFi network as shown in Table 1 . Google reported that QUIC's benefits were higher when congestion and delays were higher in the network[2]. Our observations confirm this with the results for the 3G network. However, our result conflicts with[12] where the authors found that the higher packet reordering rates in 3G network (compared to LTE) worked to QUIC's disadvantage. However, the authors also pointed out a potential problem with the two sample sets they used in their study.

4.3 | Results for the Connection-Switch Scenario

We investigated QUIC under inconsistent network conditions by the inspiration of so-called "parking lot problem." Assume that a viewer has a strong WiFi signal that enables a smooth video streaming at home or in the office. As soon as the viewer leaves the property and walks to his car, the WiFi signal fades away. When the viewer is further away from the WiFi signal, the mobile device disconnects from the active connection going through the WiFi and creates a new one(s) through the LTE or 3G mobile network. In order to simulate this scenario, we used the setup shown in Figure 5 and a script that changed the active Internet connection from WiFi to cellular or vice versa regularly at fixed intervals[1]. As soon as the player detects a connection loss within

[1]Some mobile OSes use WiFi simultaneously with LTE to cope with poor WiFi signals.

a default timeout (five seconds in our experiments), the active connection was deactivated and a new one to the HAS server over the new interface was established by the client. When the long-lived connection that had a large congestion window is lost, the client will endure a low throughput till the handshake and any slow-start like phases are completed.

As it is detailed in Table 8 , for all algorithms, the rebuffer rates were numerically reduced by 1%, and higher average playback bitrates were achieved by QUIC in the WiFi↔LTE switch scenario. Based on our observations, we confirm that QUIC increases its congestion window faster than TCP, and QUIC is able to achieve a higher throughput than TCP[12]. Hence, we argue that QUIC provides faster downloads for the segments, thus, starts the media streams more quickly.

The results are more interesting in the WiFi↔3G switch scenario. We found that BASIC algorithm had the highest rebuffer rate since it did not consider buffer occupancy in bitrate adaptation. The rebuffer rate was decreased from 13% to 6% by QUIC for this algorithm. Lower bitrates for future segments were selected by the buffer-based algorithms, thus, naturally they did not have high rebuffer rates. 1% reduction in the rebuffer rate (not as substantial as for the BASIC algorithm) was achieved by QUIC for these algorithms. A higher average playback bitrate was produced by QUIC for any of the algorithms. The results are shown in Table 9 .

4.4 | Results for the Evaluation of Multiplexing Techniques

During the tests, the available bandwidth for all types of multiplexing techniques was high enough to stream without causing a stall or rebuffer event except for the large delay and large loss scenario. The main results are highlighted below and summarized in Table 10 .

- When there was a typical delay and typical loss in the network, none of the three techniques beat the others. All techniques provided similar average playback bitrates.
- When the network had a typical delay and large loss, for HTTP/1.1, average playback bitrates increased as the number of connections increased. However, increasing the number of connections beyond eight worked to HTTP/1.1's disadvantage, and reduced the average playback bitrate. QUIC performed similar to HTTP/1.1 when HTTP/1.1 ran over two connections. When HTTP/1.1 ran over more than two connections, its performance was better than QUIC. On the other hand, QUIC achieved a higher average playback bitrate than HTTP/2 regardless of the number of parallel requests in HTTP/2.
- In the large delay and typical loss scenario, QUIC performed better than all HTTP/1.1 as well as HTTP/2 cases by selecting higher bitrates for the future segments. This result confirms that QUIC's benefits are bigger whenever delays are higher in the network. The information about the delay between when a packet was received and when the ACK was sent is carried by QUIC. The original sender achieves a better estimation of the path RTT by the help of this information.
- For the large delay and large loss scenario, HTTP/1.1 over one TCP connection showed the lowest rebuffer rate and its average playback bitrate was higher than what QUIC was able to deliver. When HTTP/1.1 ran over two or more connections, it also achieved a slightly better average playback bitrate than QUIC. However, QUIC's rebuffer rates were lower than HTTP/1.1 when the number of TCP connections was two or more. The HTTP/2 approach performed worse than the others when there is a large loss and large delay in the network. For HTTP/2, the rebuffer rate increased slightly as the number of parallel requests increased.

For HTTP/1.1 scenarios, especially when there is large delay or loss, the average playback bitrate increased as the number of connections increased up to eight connections. Multiple TCP connections can be beneficial to avoid the head-of-line blocking problem. However, each connection consumes server resources. Furthermore, using multiple connections can cause unwanted side effects in congested networks and the connection might be rejected by the server since the traffic can be deemed abusive.

HTTP/2's multiplexing feature looks not feasible to multiplex the data for HAS applications. It could not beat any other protocol in any of the scenarios. Although the payloads of the different streams are independent, all the data transmitted over the same TCP connection is delivered in order to the application. This could lead a missing packet on one stream to block the packets successfully received on other streams.

QUIC performed best in typical network conditions and also in large delay scenarios. In the typical delay and large loss scenario, QUIC is not as good as HTTP/1.1 running over more than two TCP connections. For large delay and large loss scenarios, QUIC performed worse than HTTP/1.1 over one TCP connection in terms of the rebuffer rate. However, HTTP/1.1's superiority over QUIC varies as we increase the number of TCP connections in terms of the average bitrate and rebuffer rate.

4.5 | Results for the Evaluation of Fairness

We analyze the performance of the congestion control mechanisms implemented by TCP and QUIC, and test their ability to provide fairness to all streaming clients. It is expected that QUIC and TCP should be relatively fair to each other because they both use the CUBIC congestion control algorithm.

- When there was a typical delay and typical loss in the network, we found that two competing TCP clients were fair to each other. We also observed similar behavior for two competing QUIC clients in terms of the average playback bitrate. Moreover, when a TCP and a QUIC client competed, they were fair to each other, and achieved approximately the same average playback bitrate (Figure 8).
- When the network had a typical delay and large loss, the single QUIC client performed better than the single TCP client by 37%. When two clients of the same protocol competed, they were fair to each other as they achieved a similar average playback bitrate. When the TCP and QUIC clients competed with each other, QUIC provided 40% higher average playback bitrate than TCP. The superiority of QUIC over TCP did not change significantly when the QUIC client was competing with the TCP client. In other words, QUIC utilizes the network bandwidth better than TCP regardless of it competes with another QUIC flow or a TCP flow (Figure 9).
- For the large delay and typical loss scenario, we observed similar results with respect to the typical delay and large loss scenario. Here, QUIC provided 37% higher average playback bitrate both when competing and not competing with TCP (Figure 10).
- In the large delay and large loss scenario, a single QUIC client performed worse than a single TCP client by approximately 16% in terms of the average playback bitrate. When the TCP and QUIC clients competed with each other, TCP delivered an average playback bitrate 16% higher than QUIC (Figure 11).

Our findings show that QUIC provides fairness among multiple QUIC flows in all cases. We also observe that QUIC is fair to TCP in certain cases, however, not necessarily in all cases. These findings are aligned with the conclusions of [12].

5 | CONCLUSIONS

In this study, we evaluated the performance of HAS over QUIC in uncontrolled wireless network environments in the wild. We focused on standard QoE metrics as well as the average wait time after frame seeking. QUIC empirically provided better QoE especially in terms of shorter wait times and lower rebuffer rates, and while doing so, QUIC did not decrease the average playback bitrate.

We investigated QUIC's performance when frequent IP changes occurred due to viewer mobility, especially for live video. Switching to a new network interface changes the client's IP address. TCP connections are identified by IP and port pairs whereas QUIC connections are identified by a unique Connection Identifier (CID). Using CID helps QUIC make fast switching upon network/IP changes. This may enable viewers to have a seamless transition among different networks. Client-initiated connection migration is under active development and experimentation by Google, and currently not enabled by default in Google's services. Even though the CID was not used in our tests, we saw that QUIC outperformed TCP in terms of the average playback bitrate and rebuffer rate during IP changes.

We also evaluated QUIC's performance by comparing different types of multiplexing data streams. In this evaluation, we saw that there was no advantage to QUIC for networks that had typical loss and typical delays. However, when there are long delays in the network, QUIC provided higher playback bitrates and lower rebuffer rates. Since QUIC's benefits are greater in networks that have a larger delay (but without too much loss), QUIC can help more to improve the overall viewer experience in regions where early generation 3G networks still exist.

To make the best use of QUIC's multiplexing feature, one may consider downloading subsequent segments in parallel. As most HAS applications and media decoders are only capable of processing full media segments, the streaming client has to download each segment completely before it can pass it to the decoder. However, if smaller pieces inside a segment can be passed to the decoder and the decoder is capable of decoding them, there can be further advantages. For example, the new MPEG standard, called Common Media Application Format (CMAF) has the concept of CMAF segments and CMAF fragments [43]. CMAF segments consist of one or more CMAF fragments, each of which is independently decodable. In other words, one

CMAF fragment can be decoded without the need of other CMAF fragments. In practice, this means that if a CMAF fragment cannot be fetched in time but the subsequent CMAF fragments have been already received, the streaming client might prefer to skip the missing fragment in the interest of avoiding a stall. While this results in a content skip, the viewer might still be happier compared to having to endure a complete stall. We should, however, note that this approach requires certain modifications to the content encoding/packaging as well as the bitrate adaptation schemes the streaming clients use.

We also investigated QUIC's fairness to TCP. We found that QUIC provided fairness among QUIC flows in all scenarios, however, QUIC was not necessarily always fair to TCP in our experiments. Our recommendation is that as the QUIC protocol development as well as the congestion control algorithms evolve, this issue should be revisited.

Finally, we acknowledge that QUIC is already widely deployed for most Google services such as the Chrome browser and YouTube, as well as in Google's own network. Experimental results so far including ours are encouraging and we should expect further improvements as the IETF finalizes the QUIC architecture, and QUIC becomes a front runner among the protocols used for prime time streaming.

ACKNOWLEDGMENTS

An earlier version of this study and some of the results presented in this study have been published in the Proceedings of Packet Video Workshop 2018 (DOI=10.1145/3210424.3210426).

References

1. Arisu Sevket, Yildiz Ertan, Begen Ali C.. Game of protocols: Is QUIC ready for prime time streaming?. *International Journal of Network Management.* ;.

2. Langley Adam, Riddoch Alistair, Wilk Alyssa, et al. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In: SIGCOMM '17:183–196; 2017.

3. Sandvine . *Global Internet Phenomena Report.* 2016.

4. Dobrian Florin, Sekar Vyas, Awan Asad, et al. Understanding the Impact of Video Quality on User Engagement. *SIGCOMM Comput. Commun. Rev.*. 2011;41(4):362–373.

5. Balachandran Athula, Sekar Vyas, Akella Aditya, Seshan Srinivasan, Stoica Ion, Zhang Hui. Developing a Predictive Model of Quality of Experience for Internet Video. *SIGCOMM Comput. Commun. Rev.*. 2013;43(4):339–350.

6. Arisu Sevket, Begen Ali C.. Quickly Starting Media Streams Using QUIC. In: Proceedings of the 23rd Packet Video Workshop:1–6; 2018.

7. Timmerer Christian, Bertoni Alan. Advanced Transport Options for the Dynamic Adaptive Streaming over HTTP. *Computing Research Repository.* 2016;abs/1606.00264.

8. Szabo G., Racz S., Bezzera D., Nogueira I., Sadok D.. Media QoE enhancement With QUIC. In: 2016 IEEE Conference on Computer Communications Workshops:219-220; 2016.

9. Li Bo, Wang Chengzhi, Xu Yiling, Ma Zhan. An MMT based heterogeneous multimedia system using QUIC. In: 2016 2nd International Conference on Cloud Computing and Internet of Things (CCIOT):129-133; 2016.

10. Bhat Divyashri, Rizk Amr, Zink Michael. Not So QUIC: A Performance Study of DASH over QUIC. In: Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video:13–18; 2017.

11. Zinner T., Geissler S., Helmschrott F., Burger V.. Comparison of the initial delay for video playout start for different HTTP-based transport protocols. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM):1027-1030; 2017.

12. Kakhki Arash Molavi, Jero Samuel, Choffnes David, Nita-Rotaru Cristina, Mislove Alan. Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols. In: Proceedings of the 2017 Internet Measurement Conference:290–303; 2017.

13. Ayad Ibrahim, Im Youngbin, Keller Eric, Ha Sangtae. A Practical Evaluation of Rate Adaptation Algorithms in HTTP-based Adaptive Streaming. *Computer Networks.* 2018;133:90 - 103.

14. Carlucci Gaetano, De Cicco Luca, Mascolo Saverio. HTTP over UDP: An Experimental Investigation of QUIC. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing:609–614; 2015.

15. Megyesi P., Kramer Z., Molnar S.. How quick is QUIC?. In: 2016 IEEE International Conference on Communications (ICC):1-6; 2016.

16. Cook S., Mathieu B., Truong P., Hamchaoui I.. QUIC: Better for what and for whom?. In: 2017 IEEE International Conference on Communications (ICC):1-6; 2017.

17. Qian P., Wang N., Tafazolli R.. Achieving Robust Mobile Web Content Delivery Performance Based on Multiple Coordinated QUIC Connections. *IEEE Access.* 2018;6:11313-11328.

18. Viernickel T., Froemmgen A., Rizk A., Koldehofe B., Steinmetz R.. Multipath QUIC: A Deployable Multipath Transport Protocol. In: 2018 IEEE International Conference on Communications (ICC):1-7; 2018.

19. De Coninck Quentin, Bonaventure Olivier. Multipath QUIC: Design and Evaluation. In: Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies:160–166; 2017.

20. Yang S., Li H., Wu Q.. Performance Analysis of QUIC Protocol in Integrated Satellites and Terrestrial Networks. In: 2018 14th International Wireless Communications Mobile Computing Conference (IWCMC):1425-1430; 2018.

21. Hussein Ali, Kayssi Ayman, Elhajj Imad H., Chehab Ali. SDN for QUIC: An Enhanced Architecture with Improved Connection Establishment. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing:2136–2139; 2018.

22. Yu Y., Xu M., Yang Y.. When QUIC meets TCP: An experimental study. In: 2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC):1-8; 2017.

23. Caddy QUIC support http://github.com/mholt/caddy/wiki/QUIC, online; accessed Jan. 2018.

24. quic-go issues http://github.com/lucas-clemente/quic-go/issues/302, online; accessed Jan. 2018.

25. Playing with QUIC http://www.chromium.org/quic/playing-with-quic, online; accessed Sep. 2017.

26. Juluri P., Tamarapalli V., Medhi D.. SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP. In: 2015 IEEE International Conference on Communication Workshop (ICCW):1765-1770; 2015.

27. Python urllib http://docs.python.org/2/library/urllib.html, online; accessed Dec. 2017.

28. libcurl http://curl.haxx.se/libcurl, online; accessed Oct. 2017.

29. Python sub-process https://docs.python.org/2/library/subprocess.html, online; accessed Dec. 2017.

30. GitHub quic-streaming http://github.com/sevketarisu/quic-streaming, online; accessed March. 2018.

31. Huang Te-Yuan, Johari Ramesh, McKeown Nick, Trunnell Matthew, Watson Mark. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. *SIGCOMM Comput. Commun. Rev.*. 2014;44(4):187–198.

32. Apache HTTP Server http://httpd.apache.org, online; accessed Jan. 2018.

33. tc-NetEm http://wiki.linuxfoundation.org/networking/netem, online; accessed Jan. 2018.

34. F. C. Commission. Raw Data - Measuring Broadband America. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016, online; accessed June. 2018.

35. DASH-IF Guidelines https://dashif.org/docs/DASH-AVC-264-Test-Vectors-v1.0.pdf, online; accessed June. 2018.

36. DASH Dataset http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014, online; accessed Sept. 2017.

37. Push Technology http://docs.pushtechnology.com/cloud/latest/manual/html/designguide/solution/support/connection_limitations.html, online; accessed June. 2018.

38. Fielding R., Reschke. J.. *RFC 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing.* 2014.

39. RFC 8312: CUBIC for Fast Long-Distance Networks https://tools.ietf.org/html/rfc8312, online; accessed June. 2018.

40. Bentaleb A., Begen A. C., Zimmermann R., Harous S.. SDNHAS: An SDN-Enabled Architecture to Optimize QoE in HTTP Adaptive Streaming. *IEEE Transactions on Multimedia.* 2017;19(10):2136-2151.

41. Akamai . *Maximizing audience engagement: How online video performance impacts viewer behavior?.* 2015.

42. Conviva . *OTT Streaming Market Year in Review.* 2017.

43. Begen Ali C., Syed Yasser. Are the streaming format wars over?. In: IEEE Int. Conf. Multimedia and Expo (ICME); 2018.

TABLE 1 Measured network parameters (averages).

Type	Advertised Bandwidth	Measured Tput btw. Server & Client	Average RTT	Loss Rate
WiFi	50 Mbps	5.9 Mbps	69 ms	0%
LTE	300 Mbps	5.4 Mbps	128 ms	~0%
3G	21.6 Mbps	3.1 Mbps	234 ms	~0%

TABLE 2 Frame-seek scenario for the 600-second content.

Viewer Action	Seek at	Seek to	Play Duration
Start at 0 s	-	-	40 s
Seek #1	40 s	100 s	50 s
Seek #2	150 s	200 s	80 s
Seek #3	280 s	350 s	70 s
Seek #4	420 s	500 s	100 s
Finish at 600 s	-	-	-
Total Viewed			340 s

TABLE 3 Connection-switch scenario for the 600-second content.

From Second	To Second	Connection Type
0	60	WiFi
60	180	LTE or 3G
180	300	WiFi
300	420	LTE or 3G
420	480	WiFi
480	540	LTE or 3G
540	600	WiFi

TABLE 4 Network settings for NP2b profile in controlled environment experiments.

Typical Loss & Delay			Large Loss			Large Delay			Large Delay & Loss		
Throughput (Mbps)	Delay (ms)	Loss (%)	Throughput (Mbps)	Delay (ms)	Loss (%)	Throughput (Mbps)	Delay (ms)	Loss (%)	Throughput (Mbps)	Delay (ms)	Loss (%)
1.5	100	0.12	1.5	100	0.1	1.5	200	0.12	1.5	200	0.1
2	88	0.09	2	88	1	2	400	0.09	2	400	1
3	75	0.06	3	75	2	3	600	0.06	3	600	2
4	50	0.08	4	50	5	4	800	0.08	4	800	5
5	38	0.09	5	38	10	5	1300	0.09	5	1300	10
4	50	0.08	4	50	5	4	800	0.08	4	800	5
3	75	0.06	3	75	2	3	600	0.06	3	600	2
2	88	0.09	2	88	1	2	400	0.09	2	400	1

TABLE 5 Frame-seek results with BASIC algorithm.

	Avg. (std.) Playback Bitrate (Mbps)			Avg. Wait Time after Seeking (s)			Rebuffer Rate		
	WiFi	LTE	3G	WiFi	LTE	3G	WiFi	LTE	3G
QUIC	3.38 (0.10)	3.38 (0.09)	3.37 (0.15)	1.35	1.40	1.42	1%	1%	2%
TCP	3.29 (0.23)	3.35 (0.07)	3.08 (0.17)	1.90	1.93	2.40	3%	3%	5%

TABLE 6 Frame-seek results with SARA algorithm.

	Avg. (std.) Playback Bitrate (Mbps)			Avg. Wait Time after Seeking (s)			Rebuffer Rate		
	WiFi	LTE	3G	WiFi	LTE	3G	WiFi	LTE	3G
QUIC	2.63 (0.08)	3.01 (0.10)	2.95 (0.12)	1.20	1.16	1.36	1%	1%	2%
TCP	2.57 (0.07)	2.92 (0.09)	2.87 (0.10)	2.20	2.32	2.42	4%	4%	6%

TABLE 7 Frame-seek results with BBA-2 algorithm.

	Avg. (std.) Playback Bitrate (Mbps)			Avg. Wait Time after Seeking (s)			Rebuffer Rate		
	WiFi	LTE	3G	WiFi	LTE	3G	WiFi	LTE	3G
QUIC	2.65 (0.05)	2.75 (0.07)	2.65 (0.06)	1.28	1.30	1.48	1%	2%	2%
TCP	2.53 (0.04)	2.69 (0.10)	2.62 (0.07)	2.17	2.22	2.33	4%	4%	6%

TABLE 8 WiFi-LTE switch results.

Algorithm	Protocol	Avg. (std.) Playback Bitrate	Rebuffer Rate
BASIC	QUIC	3.19 (0.05) Mbps	2%
	TCP	2.98 (0.06) Mbps	3%
SARA	QUIC	2.37 (0.04) Mbps	3%
	TCP	2.22 (0.07) Mbps	4%
BBA-2	QUIC	1.24 (0.09) Mbps	4%
	TCP	1.20 (0.08) Mbps	5%

TABLE 9 WiFi-3G switch results.

Algorithm	Protocol	Avg. (std.) Playback Bitrate	Rebuffer Rate
BASIC	QUIC	2.95 (0.07) Mbps	6%
	TCP	2.91 (0.06) Mbps	13%
SARA	QUIC	2.12 (0.08) Mbps	1%
	TCP	1.95 (0.06) Mbps	2%
BBA-2	QUIC	1.16 (0.06) Mbps	4%
	TCP	1.13 (0.04) Mbps	5%

TABLE 10 Avg. (std.) results for BASIC, SARA and BBA2 algorithms as well as DASH-IF NP2b and FCC throughput profiles.

	Typical Delay and Typical Loss		Typical Delay and Large Loss		Large Delay and Typical Loss		Large Delay and Large Loss	
	Avg. Playback Bitrate (Mbps)	Rebuffer Rate	Avg. Playback Bitrate (Mbps)	Rebuffer Rate	Avg. Playback Bitrate (Mbps)	Rebuffer Rate	Avg. Playback Bitrate (Mbps)	Rebuffer Rate
HTTP/1.1 (1 TCP)	1.94 (0.04)	0%	1.01 (0.05)	0%	0.75 (0.06)	0%	0.37 (0.04)	1%
HTTP/1.1 (2 TCPs)	1.92 (0.06)	0%	1.37 (0.07)	0%	0.77 (0.07)	0%	0.34 (0.03)	10%
HTTP/1.1 (4 TCPs)	1.91 (0.05)	0%	1.56 (0.07)	0%	0.77 (0.07)	0%	0.39 (0.03)	9%
HTTP/1.1 (6 TCPs)	1.90 (0.10)	0%	1.68 (0.04)	0%	0.80 (0.09)	0%	0.41 (0.02)	4%
HTTP/1.1 (8 TCPs)	1.91 (0.03)	0%	1.71 (0.03)	0%	0.80 (0.06)	0%	0.43 (0.03)	6%
HTTP/1.1 (10 TCPs)	1.89 (0.04)	0%	1.69 (0.05)	0%	0.79 (0.05)	0%	0.44 (0.04)	6%
HTTP/1.1 (12 TCPs)	1.90 (0.04)	0%	1.65 (0.09)	0%	0.74 (0.04)	0%	0.39 (0.03)	10%
HTTP/2 (2 Parallel)	1.85 (0.08)	0%	0.95 (0.06)	0%	0.72 (0.06)	0%	0.28 (0.03)	4%
HTTP/2 (4 Parallel)	1.84 (0.06)	0%	0.96 (0.07)	0%	0.63 (0.07)	0%	0.26 (0.02)	6%
HTTP/2 (6 Parallel)	1.92 (0.07)	0%	0.94 (0.07)	0%	0.57 (0.08)	0%	0.24 (0.02)	8%
QUIC	1.94 (0.05)	0%	1.39 (0.05)	0%	1.04 (0.07)	0%	0.32 (0.02)	4%

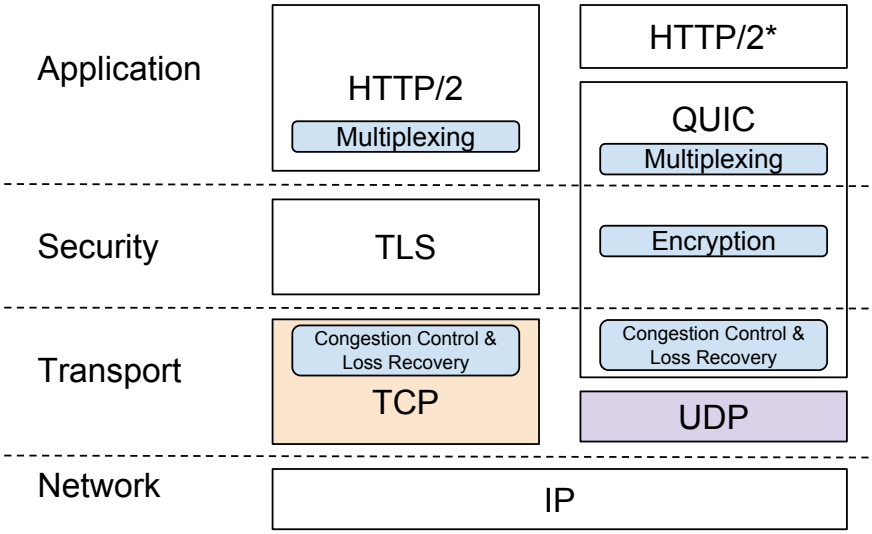

FIGURE 1 QUIC architecture (*: HTTP/2 shim).

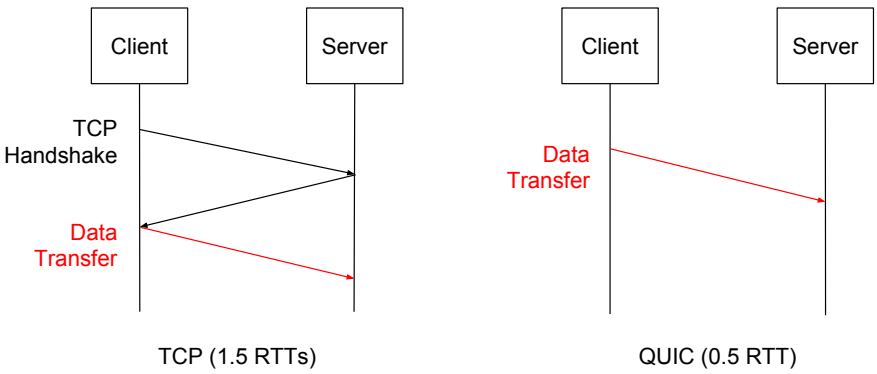

FIGURE 2 In the subsequent connections, QUIC requires 0.5 RTT while TCP requires 1.5 RTTs before any data request is received at the server.

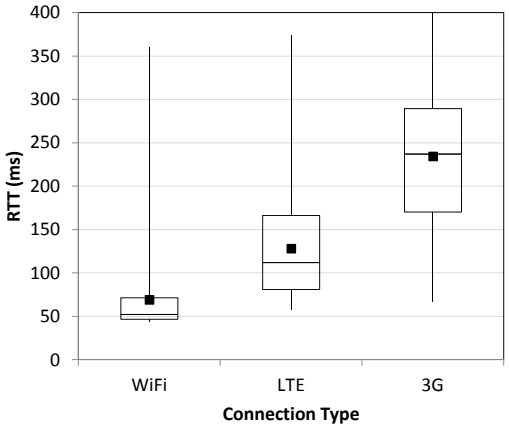

FIGURE 3 Measured RTTs during the tests (milliseconds).

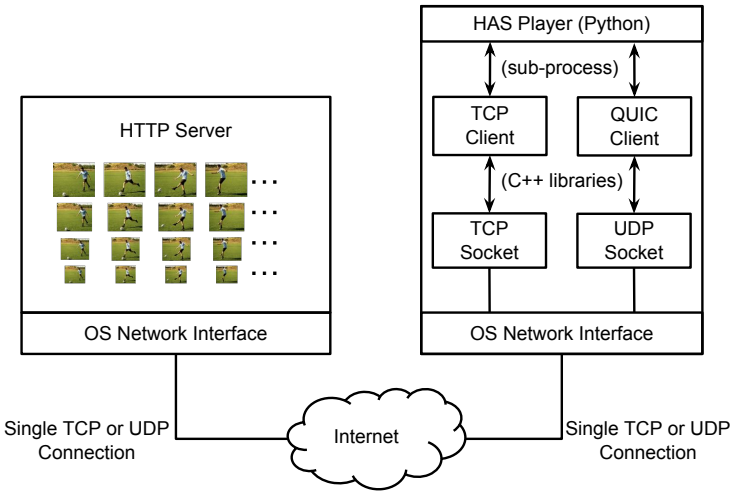

FIGURE 4 The HAS player can use QUIC or TCP to download the media segments.

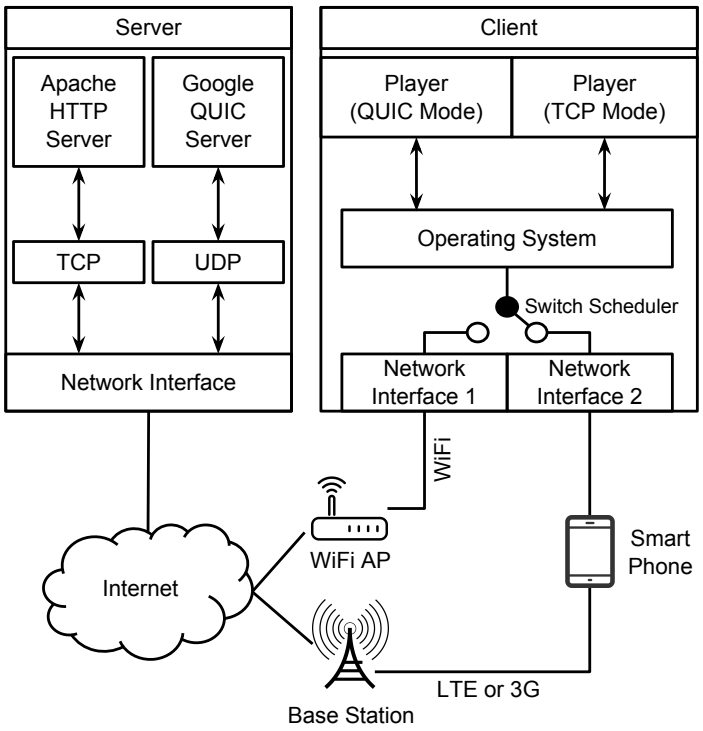

FIGURE 5 Internet setup for the connection-switch tests.

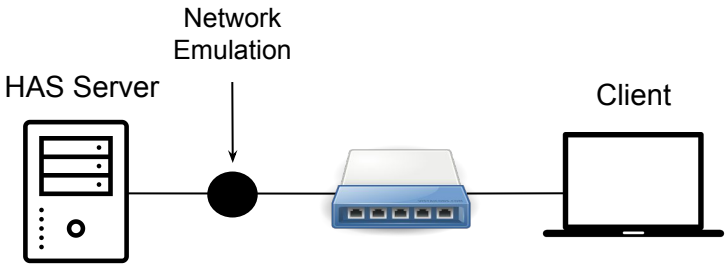

FIGURE 6 Testbed setup for the controlled experiments.

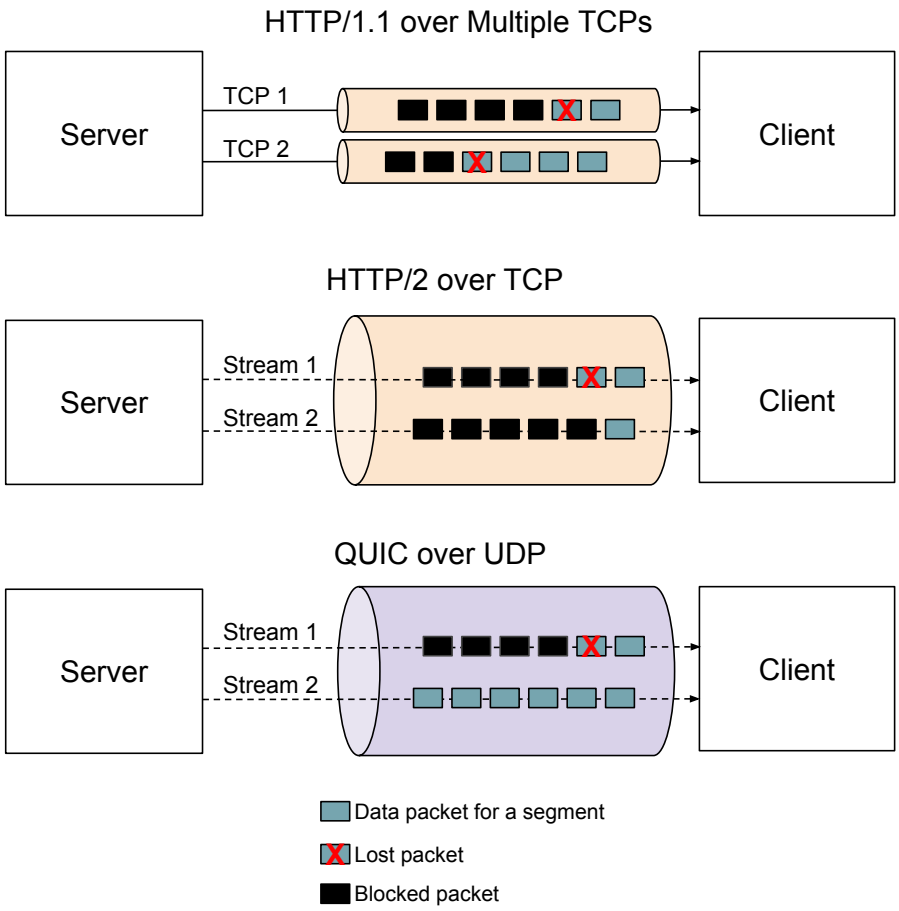

FIGURE 7 Sending multiple streams of data over multiple TCP connections (the example here is for two TCPs), or a single UDP/TCP connection.

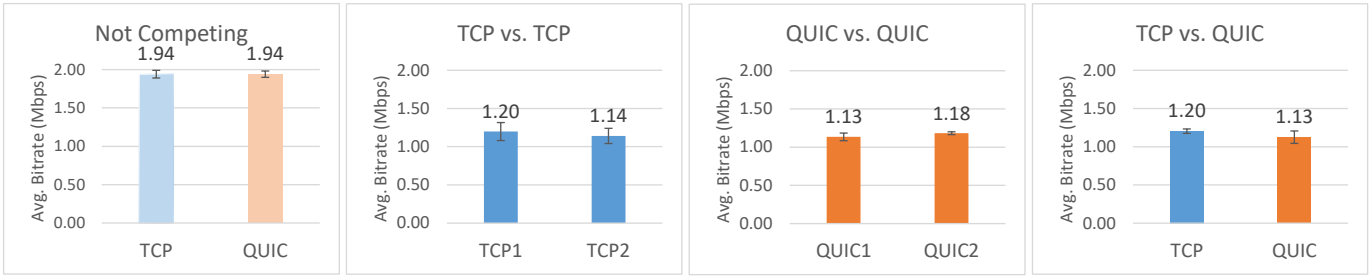

FIGURE 8 Fairness results for the Typical Delay and Typical Loss profile.

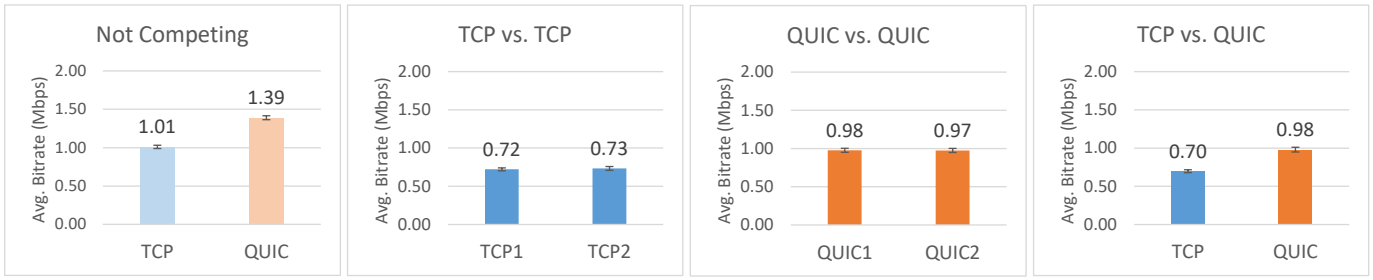

FIGURE 9 Fairness results for the Typical-Delay and Large-Loss profile.

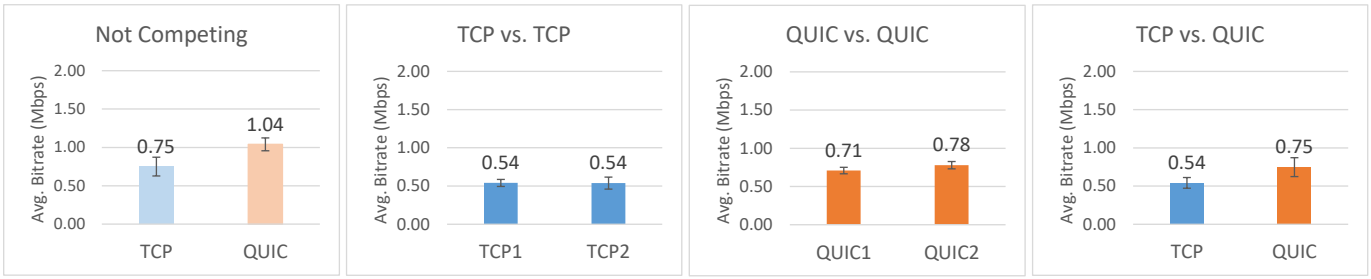

FIGURE 10 Fairness results for the Large-Delay and Typical-Loss profile.

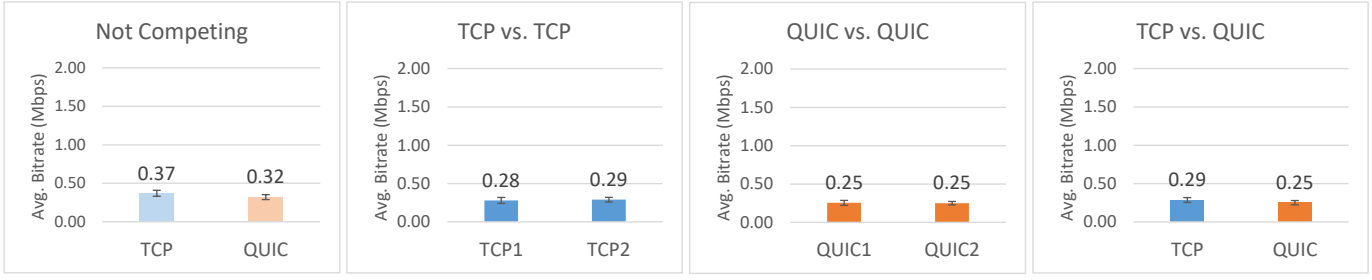

FIGURE 11 Fairness results for the Large-Delay and Large-Loss profile.