Quickly Starting Media Streams Using QUIC

Sevket Arisu
Ozyegin University and Turkcell
Istanbul, Turkey
sevket.arisu@ozu.edu.tr

Ali C. Begen
Ozyegin University
Istanbul, Turkey
ali.begen@ozyegin.edu.tr

ABSTRACT

Originally proposed by Google, QUIC is a low-latency transport protocol currently being developed and specified in the IETF. QUIC's low-latency, improved congestion control, multiplexing features are promising and may help improve viewer experience in HTTP adaptive streaming applications. To investigate what issues due to running HTTP over TCP can be alleviated by using HTTP over QUIC, we measured QUIC's streaming performance on wireless and cellular networks. Specifically, we examined QUIC's performance during network interface changes due to viewer's mobility and under unstable network conditions. Results show that QUIC starts media streams more quickly, providing a better streaming and seeking experience, in particular, when there is more congestion in the network, and outperforms TCP when the viewer is mobile and switches between the networks.

CCS CONCEPTS

• **Information systems → Information systems applications**;
• **Multimedia information systems** → *Multimedia streaming*;

KEYWORDS

HTTP adaptive streaming, DASH, QUIC, QoE.

ACM Reference Format:
Sevket Arisu and Ali C. Begen. 2018. Quickly Starting Media Streams Using QUIC. In *Packet Video'18: 23rd Packet Video Workshop, June 12–15, 2018, Amsterdam, Netherlands.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3210424.3210426

1 INTRODUCTION

Google's experimental QUIC protocol has been developed rapidly and is now deployed and working behind many Google's services at a large scale. It currently accounts for over 30% of Google's total egress traffic [21]. On the client side, QUIC is widely deployed through the Chrome browser (both desktop and mobile), and already comprises 7% of all Internet traffic [25]. QUIC provides new features such as reduced connection establishment latency, improved congestion control, multiplexing streams and encryption at application transport level. Some of these features may be changed 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.

A common misconception is that HTTP must use TCP. While that has been primarily the case, the HTTP specifications (RFCs 7230 and 7540) do not mandate the use of TCP, rather they state that HTTP requires a reliable transport protocol. To that effect, HTTP adaptive streaming (HAS) has dominantly used TCP for many years. A major drawback in TCP is the head-of-line blocking. If a packet is lost, TCP holds on to the subsequent packets that may have already been received in the receive buffer, and it does not deliver them to the application until the missing packet is recovered using retransmission in order to guarantee in-order delivery.

Regarding HAS, head-of-line blocking and slow retransmissions could occasionally lead to late delivery of the media segments and this may degrade the viewer's quality of experience (QoE) especially if the streaming client runs out of data in its playback buffer. Thanks to its design, QUIC can help with this issue. QUIC uses multiplexed streams on one UDP connection to handle multiple requests in parallel. Each stream in QUIC has reliable delivery, but they are independent of each other. If a packet gets lost in a stream, the other streams are not affected. HTTP/2 (RFC 7540) also has multiplexing features, however, HTTP/2 may still suffer from head-of-line blocking if it is used over TCP. QUIC's another feature that can help improve QoE is the reduced handshake latency. TCP requires 1.5 round-trip times (RTT), whereas QUIC requires half an RTT before any data request is received by the server. Potentially, this may reduce the initial startup or seeking latency. Lastly, QUIC provides improved congestion control and loss recovery features, which may help achieve a higher throughput in general.

Previous research has confirmed that viewers are very sensitive to buffering as one percent increase in buffering can cause reduction in viewing time more than three minutes [17]. According to another research [11], engagement linearly decreases with increasing rate of buffering up to a certain threshold. Only 30% of the content is watched when there are more than 0.3 buffering events per minute. If the rate of buffering increases further, viewers get annoyed and they quit early without finishing to play the entire content. Buffering events occur not only when there is a loss or delay in the network during playback, but also when the viewer seeks to a point in time and the corresponding media segments have not been buffered yet. Furthermore, buffering events are more likely to occur when the network interface changes since in this case the TCP connection has to be re-established due to the change of the IP address (*e.g.*, switching from WiFi to LTE or 3G network).

The goal of this paper is to answer the following two questions: First, how and under what circumstances can QUIC start media streams more quickly, helping to reduce the initial startup and seeking latency? Second, how can we benefit from QUIC to get better QoE when the viewer is mobile and switches between the networks? We evaluate the HAS performance over QUIC vs. over TCP to answer these questions.

This paper continues as follows: In Section 2, we present the previous works that studied 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 scenarios and varying network conditions. In Section 4, we analyze the observed QoE measurements. Finally, Section 5 includes our conclusions, and a discussion of potential benefits of QUIC transport for HAS and future work.

2 RELATED WORK

Timmerer *et al.* found that using QUIC instead of TCP did not impact the overall streaming performance at the client in terms of increased or decreased media throughput [27]. On the other hand, Szabo *et al.* showed QUIC's gain in initial buffering time to be in the range of 6-49% depending on the video properties and network environment [26]. Li *et al.* studied an MMT-based heterogeneous multimedia system using QUIC. Their finding is that QUIC is a better choice for media transport in comparison with HTTP when using an MMT system [22]. Bhat *et al.* evaluated the performance of the state-of-the-art HAS algorithms by comparing TCP versus QUIC at the transport layer. The authors found that QUIC did not benefit the existing algorithms because these algorithms were designed with TCP in mind [13]. Zinner *et al.* also looked into the same problem and found for the playback start, QUIC with 0-RTT connection establishment clearly performed better than the other protocols [29]. Google reported that QUIC reduced rebuffer rates of YouTube playbacks by 18% for desktop users and 15.3% for mobile users [21]. In their study, Kakhki *et al.* found that QUIC provided better streaming QoE, but only for high-quality video [20]. Ayad *et al.* studied the performance of commercial and open-source players and found that the QUIC protocol was quite aggressive when competing with other TCP flows and not as responsive to congestion as other TCP flows [10].

There are also studies that investigated QUIC for not necessarily streaming over HTTP but ordinary Web transport. Carlucci *et al.* found that QUIC outperformed HTTP over TCP in terms of page load times when there were no random losses and QUIC outperformed SPDY (the pre-standard version of HTTP/2) in the case of a lossy connection [14]. Megyesi *et al.* compared QUIC's Web page load performance in HTTP/1.1 and SPDY, and concluded that none of these protocols is clearly better than the other two and the actual network conditions determined which protocol performed better [23]. Cook *et al.* clearly saw that QUIC outperformed HTTP/2 over TCP/TLS in unstable networks such as wireless mobile networks [16]. In a recent work, Qian *et al.* studied mobile Web content delivery and found that multiple QUIC connections could overcome the limitations of different congestion control algorithms when downloading short-lived content [24].

Some research found that QUIC was less competitive than TCP in a network with little loss, large buffer or large propagation delay [28] or QUIC did not provide a significant boost to adaptive streaming performance [13, 27]. We suspect that the contradictory nature of these results (compared to ours) is primarily due to the fact that these studies used either an older version of the QUIC server provided by Google or an open-source implementation, which was not provided by Google. Since the QUIC code evolves rapidly, an older or non-original code base is not guaranteed to deliver the true protocol performance. Google states that the public versions of the QUIC server and client are not "performant at scale" [7], however, our testing showed that the performances of the toy server and the toy client were good enough for testing with one client.

To the best of our knowledge, we are the first to investigate QUIC with respect to viewer's frame-seek requests and frequent wireless network changes over the public Internet. Our differences compared to prior work are listed below and summarized in Table 1:

- We used the latest official public version of the QUIC server.
- We tested frame-seek scenarios to understand how much quicker QUIC was compared to TCP.
- We tested scenarios involving connection/interface changes.
- We experimented with live video content.
- We primarily focused on the wireless environments.

3 APPROACH AND ENVIRONMENT SETUP

In this section, we describe our test setup, player modifications and approach to measure the performance of HAS over QUIC and TCP for different frame-seek scenarios and network interface changes. In this study, we present results only for forward frame-seek scenarios, however, the results and conclusions are equally applicable for backward frame-seek scenarios as well.

We ran the experiments on the public Internet (uncontrolled environment). We did not use any test bed or any traffic shaping tool to modify the network speed, loss or delay. We used the official QUIC server (v39) provided by Google [7] for QUIC and Apache's HTTP server [2] for TCP. The servers were set up on an Amazon EC2 instance located in Frankfurt (Germany), while the streaming clients were located in Istanbul (Turkey). The streaming clients were connected via WiFi to residential broadband access network or via smartphone tethering to a commercial LTE or 3G network.

Figure 1 and Table 2 show the network characteristics observed during the tests. The lower, middle and upper bars in the box plots in Figure 1 represent the 25^{th}, 50^{th} and 75^{th} percentiles of the measured RTTs, respectively, for all three types of wireless networks. The average RTTs (shown as black squares) for WiFi, LTE and 3G are 69 ms, 128 ms and 234 ms, respectively. Note that the RTTs that were higher than 400 ms for 3G are omitted in Figure 1.

We made a set of modifications in the Python-based player [19] to ensure a fair comparison of QUIC and TCP. First, because of the lack of a Python library implementation for QUIC, we integrated the QUIC client that was provided by Google into the player as a sub-process. The media segments were downloaded by this sub-process using a single QUIC connection over UDP. Second, the original player [19] used Python's *urllib* [1] to make HTTP requests, and the player was opening a new connection for each segment. However, QUIC made transfers over a single UDP connection by default. To make the comparison fair, we integrated a new TCP client and used this client instead of Python's *urllib* to download the segments over a single TCP connection. The TCP client is built with *libcurl* [6] and works with HTTP keep-alive feature on. Figure 2 shows how the HAS player works with QUIC and TCP.

Table 1: Comparison between our and prior work (sorted by publication date).

	Tested QUIC Version	Used Official Google Server?	Wireless Networks[1]	Tested Different Algorithms?	Evaluated Frame Seeking?	Evaluated Conn. Switches?	Tested Live Video?
Timmerer [27]	v19	✓	✗	✗	✗	✗	✗
Szabo [26]	Latest [2]	✗	Only WiFi	✗	✗	✗	✗
Li [22]	Latest [3]	✓	✗	✗	✗	✗	✗
Bhat [13]	Latest [2]	✗	Only WiFi	✓	✗	✗	✗
Zinner [29]	Latest [3]	✓	✗	✗	✗	✗	✗
Kakhki [20]	v37	✓	All	✗	✗	✗	✗
Ayad [10]	Latest [3]	✓	✗	✓	✗	✗	✗
Our work	v39 [3]	✓	All	✓	✓	✓	✓

[1] WiFi, 4G/LTE and 3G.
[2] Based on the third-party implementation version at the time of research.
[3] Latest at the time of research.

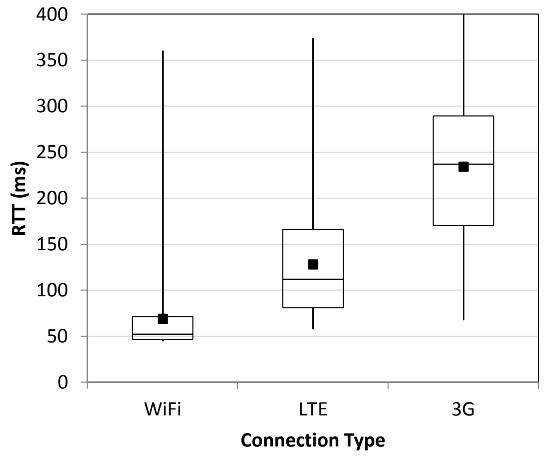

Figure 1: Measured RTTs during the tests (milliseconds).

Table 2: Measured network parameters (averages).

Type	Advertised Bandwidth	Measured Tput btw. Server & Client	Average RTT	Loss Rate
WiFi	50 Mbps	3.9 Mbps	69 ms	0%
LTE	300 Mbps	2.4 Mbps	128 ms	~0%
3G	21.6 Mbps	1.9 Mbps	234 ms	~0%

Both QUIC and TCP clients were coded in C++, and we disabled the QUIC server's in-memory cache. The modified player code, the QUIC and TCP clients are available for public access on GitHub for the research community [5]. In the future, we plan to integrate HTTP/2 support into our code. For this study, we used the following bitrate 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 measured average throughput [19].
- **SARA - Segment Aware (Buffer based):** SARA considers the segment size variation in addition to the estimated bandwidth and the current buffer occupancy to accurately predict the time required to download the next segment. SARA estimates throughput with weighted harmonic mean [19].
- **BBA-2 (Buffer based):** BBA-2 algorithm uses a set of functions that maps the buffer occupancy to a bitrate. It tries to reduce the rebuffering and increase the average playback bitrate. It directly chooses the next bitrate based on the buffer occupancy and only uses bandwidth estimation when necessary. BBA-2 algorithm was part of a large-scale Netflix experiment [18].

Note that this study primarily focuses on the transport options for HAS, not the features of the particular bitrate adaptation algorithms.

3.1 Frame-Seek Scenario

To evaluate the performance of QUIC and TCP when the viewer wanted to perform seeking, we implemented the frame-seek feature in the player. Upon seeking to a point in time where the

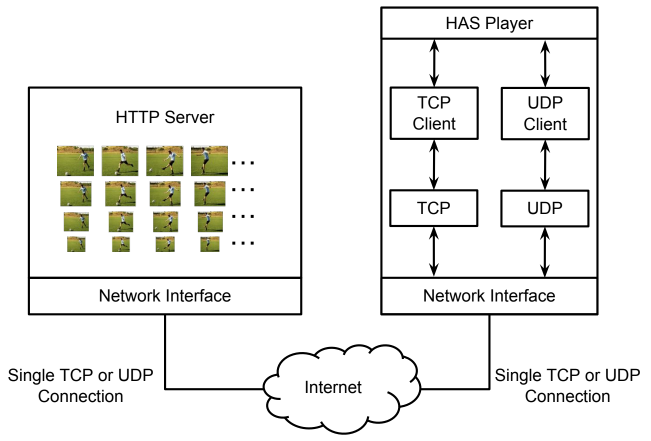

Figure 2: The HAS player can use QUIC or TCP to download the media segments.

corresponding media segments have not been buffered yet, the playback buffer will be empty and the player needs to fill it up quickly for a smooth seeking experience. The frame-seek scenario is tabulated in Table 3, which shows that there are four seeking events at various seek-at seconds. When the media time equals a seek-at time, the player jumps to the seek-to time. In conventional adaptive streaming, the bitrate adaptation algorithm may decide to fetch the segments at the lowest encoding bitrate to reduce the seeking time at the expense of reduced presentation quality. In our tests, we programmed the player to keep the bitrate the same with the one of the last played segment before the seeking, and measured the time difference from the request to the playback of the seek-to segment. While there is a clear trade-off between the seeking time and seeking quality, our goal was to see the impact of the transport layer on the seeking time, and thus, we kept the requested bitrate unchanged during seeking. We used an on-demand content in the frame-seek scenario tests.

Table 3: 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

3.2 Connection-Switch Scenario

To evaluate QUIC's performance against varying network conditions and IP connections/disconnections, we placed a network interface switcher script at the client machine. This script changed the client's active network connection from WiFi to LTE or 3G or vice versa within fixed intervals. When a connection loss was detected by the player, it tried to restore the connection by using a new interface. During this process, the playback continued as long as there were media segment(s) in the playback buffer. Naturally, the connection re-establishment time may result in a stall, increase rebuffering and degrade the viewer QoE. We measured the average playback bitrate and rebuffer rate when there were various IP change events that were caused by network interface changes.

Today, thanks to the networking stack developments, most connections can be re-established in several seconds. If the player is streaming an on-demand content with a sufficiently large playback buffer size, the player can absorb most connection switches without resulting in a stall or significant quality degradations. However, if the connection restores in a longer time, the player may have to fetch segments encoded at lower bitrates. If this does not suffice, a stall and rebuffering is inevitable.

To better understand the impact of using QUIC vs. TCP during connection switches, we used a live content with a smaller playback buffer size. The setup for these tests is shown in Figure 3, and the connection-switch scenario is tabulated in Table 4.

4 RESULTS

In our tests, we used 600-second long video dataset from [8] with 20 bitrates ranging from 44 Kbps to 3.9 Mbps. We ran the tests for three

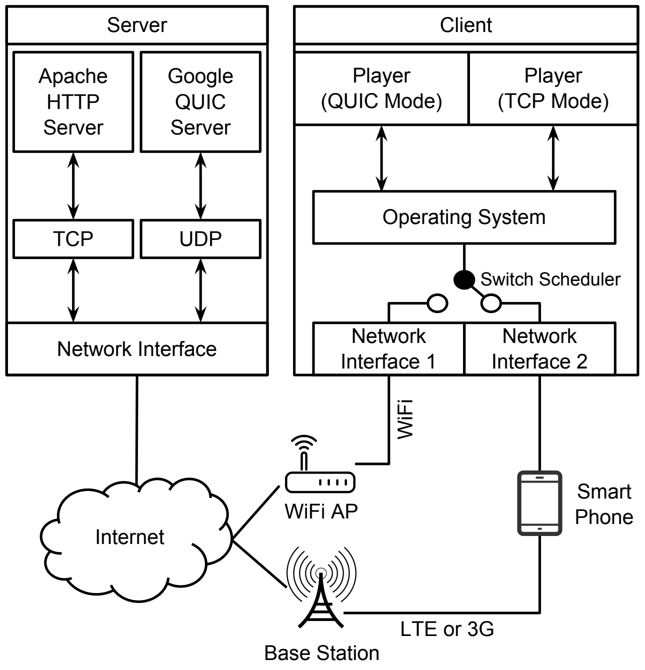

Figure 3: Internet setup for the connection-switch tests.

Table 4: 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

different algorithms on three types of wireless networks in separate time slots. However, we started the players for each protocol (QUIC and TCP) approximately at the same time to ensure both protocols faced the same conditions in the uncontrolled environment. We also repeated each test for 10 times to avoid significant anomalies and in this section, we present the average results. 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.

4.1 Metrics

We measured following metrics in our tests:

- *Average Playback Bitrate*: We measured the average bitrates of the downloaded segments 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 [12].

- *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 [9, 15].
- *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.

In our tests, the players using QUIC always achieved a higher average playback bitrate by selecting segments encoded at bitrates higher than, or at least as high as, TCP while not increasing, and actually reducing, the average wait time or rebuffer rate. The results are presented in Tables 5, 6 and 7.

4.2 Results for the Frame-Seek Scenario

The middle columns in Tables 5, 6 and 7 show the frame-seek results for the BASIC, SARA and BBA-2 algorithms, respectively. In all cases, QUIC achieved a shorter average wait time after seeking (up to 50% reduction compared to the wait times achieved by TCP), providing a quicker start of the media streams. The rebuffer rates were also substantially lower for QUIC. These results may seem to conflict with some of the results reported by earlier studies [13]. We suspect this is primarily due to the fact that [13] used an open-source server implementation with experimental QUIC support [3, 4].

The HAS player may have to flush the entire playback buffer upon a frame seek. The SARA and BBA-2 algorithms are sensitive to empty buffer, while the BASIC algorithm does not consider the current buffer level to determine the next bitrate. During the tests, the available bandwidth on WiFi was high enough to stream near the highest encoding bitrate. For this reason, the BASIC algorithm gives a better average playback bitrate than the other two algorithms. The buffer-based algorithms may be modified for the frame-seek feature to better compete with the throughput-based algorithms.

We observed that the average wait time after seeking is almost two seconds or longer in all cases when streaming over TCP for all algorithms on all types of wireless networks. QUIC reduced the average wait time to less than one and a half second, which is an important result, considering that keeping the average wait time after seeking less than two seconds is essential for viewer engagement and loyalty [9, 15]. We also observed that QUIC reduced the rebuffer rates. 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.

In our measurements, the 3G network has 82% higher average RTT than the LTE network as shown in Table 2. According to Google, QUIC's benefits are bigger whenever congestion, packet loss and delays are higher in the network [21]. Our findings confirm this with the results for the 3G network. However, our findings conflict with [20] where the authors found that 3G network packet

Table 5: Frame-seek results with BASIC algorithm.

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

Table 6: Frame-seek results with SARA algorithm.

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

Table 7: Frame-seek results with BBA-2 algorithm.

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

reordering rates were higher compared to LTE and this 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

Inspired by the so-called "parking lot problem", we looked into using QUIC under unstable network conditions. The use case is that the viewer has a strong WiFi signal and has been streaming video at home (or in the office), and then the viewer leaves the premises, walks to his car, and the WiFi signal fades away. After a certain amount of time, the mobile device disconnects any connection going through the WiFi and creates a new one(s) through the LTE or 3G mobile network. We used the setup shown in Figure 3 for simulating this scenario. We placed a script on the client machine that changed the active Internet connection from WiFi to cellular or vice versa periodically at fixed intervals[1]. When the player detected a connection loss within a predefined timeout (five seconds in our tests), it killed the active connection and re-established a new one to the HAS server over the new interface. Once the long-lived connection that had a large congestion window is lost, the player will endure a low throughput till the handshake and any slow-start like phase are completed.

In the WiFi↔ LTE switch scenario, QUIC reduced the rebuffer rates by 1% and provided higher average playback bitrates for all algorithms, as detailed in Table 8. Our observations confirm that QUIC increases its window more aggressively than TCP and QUIC is able to achieve a larger congestion window when competing with

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

Table 8: WiFi-LTE switch results.

Algorithm	Protocol	Avg. Playback Bitrate	Rebuffer Rate
BASIC	QUIC	3.19 Mbps	2%
	TCP	2.98 Mbps	3%
SARA	QUIC	2.37 Mbps	3%
	TCP	2.22 Mbps	4%
BBA-2	QUIC	1.24 Mbps	4%
	TCP	1.20 Mbps	5%

Table 9: WiFi-3G switch results.

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

TCP [20]. Thus, QUIC provides faster downloads for the segments, starts the media streams more quickly and reduces the rebuffer rate soon after the connection establishment.

In the WiFi↔3G switch scenario, the results are more interesting. Since the BASIC algorithm did not consider buffer occupancy in bitrate adaptation, it had the highest rebuffer rate. For this algorithm, QUIC reduced the rebuffer rate from 13% to 6%. Buffer-based algorithms already did not have high rebuffer rates since they selected lower bitrates for future segments. For these algorithms, the benefit of QUIC was 1% reduction in the rebuffer rate (not as substantial as for the BASIC algorithm). For any of the algorithms, QUIC still delivered a higher average playback bitrate. The results are shown in Table 9.

5 CONCLUSIONS

In this paper, we evaluated the performance of HAS over QUIC on uncontrolled wireless network environments in the public Internet. 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 cause a decrease in the average playback bitrate.

We also investigated QUIC's performance when frequent IP changes existed due to viewer mobility, especially for live video. Switching to a new network interface changes the client IP address. TCP connections are identified by IP and port pairs whereas QUIC connections are identified by unique Connection Identifier (CID). Using CID may help QUIC make fast switching upon network/IP changes. Even when the CID was not used in our tests, we saw that QUIC outperformed TCP.

Since QUIC's benefits are greater in networks that have larger delay and loss, QUIC can help more to improve the overall viewer QoE in regions where early generation 3G networks still exist. However, we should note that QUIC is currently being specified in the IETF and some architectural changes are planned. Should there be significant changes in the protocol, some of the tests will need to be repeated.

REFERENCES

[1] Python urllib. http://docs.python.org/2/library/urllib.html, accessed Dec. 2017.
[2] Apache HTTP server. http://httpd.apache.org, accessed Jan. 2018.
[3] Caddy QUIC support. http://github.com/mholt/caddy/wiki/QUIC, accessed Jan. 2018.
[4] quic-go issues. http://github.com/lucas-clemente/quic-go/issues/302, accessed Jan. 2018.
[5] GitHub quic-streaming. http://github.com/sevketarisu/quic-streaming, accessed March. 2018.
[6] libcurl. http://curl.haxx.se/libcurl, accessed Oct. 2017.
[7] Playing with QUIC. http://www.chromium.org/quic/playing-with-quic, accessed Sep. 2017.
[8] DASH dataset. http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014, accessed Sept. 2017.
[9] Akamai. Maximizing audience engagement: How online video performance impacts viewer behavior. 2015.
[10] I. Ayad, Y. Im, E. Keller, and S. Ha. A practical evaluation of rate adaptation algorithms in HTTP-based adaptive streaming. *Computer Networks*, 133:90 – 103, 2018.
[11] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. Developing a predictive model of quality of experience for Internet video. *SIGCOMM Comput. Commun. Rev.*, 43(4):339–350, 2013.
[12] A. Bentaleb, A. C. Begen, R. Zimmermann, and S. Harous. Sdnhas: An SDN-enabled architecture to optimize QoE in HTTP adaptive streaming. *IEEE Transactions on Multimedia*, 19(10):2136–2151, 2017.
[13] D. Bhat, A. Rizk, and M. Zink. 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*, 2017.
[14] G. Carlucci, L. De Cicco, and S. Mascolo. HTTP over UDP: An experimental investigation of QUIC. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015.
[15] Conviva. OTT streaming market year in review. 2017.
[16] S. Cook, B. Mathieu, P. Truong, and I. Hamchaoui. QUIC: Better for what and for whom? In *2017 IEEE International Conference on Communications (ICC)*, 2017.
[17] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. *SIGCOMM Comput. Commun. Rev.*, 41(4):362–373, 2011.
[18] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, 2014.
[19] P. Juluri, V. Tamarapalli, and D. Medhi. SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP. In *2015 IEEE International Conference on Communication Workshop (ICCW)*, 2015.
[20] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove. Taking a long look at QUIC: An approach for rigorous evaluation of rapidly evolving transport protocols. In *Proceedings of the 2017 Internet Measurement Conference*, 2017.
[21] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi. The QUIC transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017.
[22] B. Li, C. Wang, Y. Xu, and Z. Ma. An MMT based heterogeneous multimedia system using QUIC. In *2016 2nd International Conference on Cloud Computing and Internet of Things (CCIOT)*, 2016.
[23] P. Megyesi, Z. Kramer, and S. Molnar. How quick is QUIC? In *2016 IEEE International Conference on Communications (ICC)*, 2016.
[24] P. Qian, N. Wang, and R. Tafazolli. Achieving robust mobile Web content delivery performance based on multiple coordinated QUIC connections. *IEEE Access*, 6:11313–11328, 2018.
[25] Sandvine. Global Internet Phenomena Report. 2016.
[26] G. Szabo, S. Racz, D. Bezzera, I. Nogueira, and D. Sadok. Media QoE enhancement with QUIC. In *2016 IEEE Conference on Computer Communications (INFOCOM) Workshops*, 2016.
[27] C. Timmerer and A. Bertoni. Advanced transport options for the dynamic adaptive streaming over HTTP. *CoRR*, abs/1606.00264, 2016.
[28] Y. Yu, M. Xu, and Y. Yang. When QUIC meets TCP: An experimental study. In *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, 2017.
[29] T. Zinner, S. Geissler, F. Helmschrott, and V. Burger. 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)*, 2017.