HTTP ADAPTIVE STREAMING WITH ADVANCED TRANSPORT

A Thesis

by

Şevket Arısu

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master Of Science

in the
Department of Computer Science

Özyeğin University
September 2018

HTTP ADAPTIVE STREAMING WITH ADVANCED TRANSPORT

Approved by:

Asst. Prof. Ali Cengiz Beğen, Advisor
Department of Computer Science
Özyeğin University

Prof. Dr. Reha Civanlar
Department of Computer Science
Özyeğin University

Asst. Prof. Tufan Coşkun Karalar
Electronics and Communication Engineering
Istanbul Technical University

Date Approved: 9 August 2018

To my wife Irmak

ABSTRACT

QUIC (Quick UDP Internet Connections) is an experimental and low-latency transport network 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. In this master thesis, 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 investigated 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.

ÖZETÇE

QUIC (Quick UDP Internet Connections - Hızlı UDP Internet Bağlantıları) Google tarafından bulunan ve halen IETF bünyesinde belirlenmekte ve geliştirilmekte olan, düşük gecikmeli ve deneysel bir ağ haberleşme protokolüdür. HTTP üzerinden dinamik adaptif iletim (DASH - Dynamic Adaptive Streaming over HTTP) uygulamalarındaki kullanıcı deneyimi, QUIC'in sunduğu düşük gecikme, gelişmiş trafik kontrolü ve çoğullamalı kanallar özellikleri sayesinde geliştirilebilir. Bunu araştırmak amacıyla, TCP üzerinden HTTP kullanıldığında oluşan problemlerin, QUIC üzerinden HTTP kullanılarak azaltılıp azaltılmadığını anlamak için kablosuz ve hücresel ağlar üzerinde QUIC'in iletim performansını ölçtük. QUIC'in performansını izleyicinin hareket halinde olmasından kaynaklanan bağlantı arayüzü değişikliklerinin bulunduğu ağ koşullarında gözlemledik. Bu araştırmalarımız sonucunda, QUIC'in medyaları daha hızlı başlattığını, özellikle yüksek trafik sıkışıklığı olduğunda daha iyi izleme ve görüntü arama deneyimi sunduğunu ve kullanıcının hareketli olduğu ve kablosuz ağlar arasında bağlantı değiştirdiği durumlarda TCP'den daha iyi performans gösterdiğini gözlemledik. Buna ek olarak, çeşitli oranlarda gecikme ve kayıpların olduğu deneysel bir ağ ortamında, QUIC'in çoğullamalı kanallar özelliğinin, HTTP üzerinden adaptif iletim için nasıl faydalı olabileceğini araştırdık. Çoklu TCP bağlantıları üzerinden HTTP1.1, tek TCP bağlantısı üzerinden HTTP/2 ve tek UDP bağlantısı üzerinden QUIC ile çoğullamalı kanallar kullanan adaptif iletim uygulamalarının performanslarını karşılaştırdık. Bu hususta, QUIC'in yüksek gecikme olan ağlarda TCP'den daha iyi performans gösterdiğini gözlemledik. Buna karşın, QUIC'in yüksek kayıp olan ağlarda önemli bir iyileşme sağlamadığını gördük.

ACKNOWLEDGEMENTS

I would like to acknowledge Asst. Prof. Ali Cengiz Beğen for giving me the opportunity to realize this thesis and for his highly valuable comments and advice, which have contributed immensely to this master thesis.

I would like to thank my teachers, particularly Princess and Birsen, for their support. I am greatly thankful to my colleagues at work especially Ertan and Erkan, for their advice and for sharing their experience.

Special thanks go to my lovely wife Irmak for her immense support during my studies. Finally, I would like to express my infinite gratitude to my sisters, my mother and my father for their outstanding support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Today, the QUIC protocol is distributed and extensively being used in many Google's services. Over 30% of Google's total egress traffic is QUIC [1]. QUIC is widely deployed to the user space by the Chrome browser (both desktop and mobile), and already comprises 7% of all Internet traffic [2]. 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.

1.1 Media Stream Delivery

Media stream delivery can be explained as providing a content to a viewer through a network with respect to media constraints and network conditions. Media content constraints are related to the fact that streams have timing constraints within one stream and as well as within different streams. Various different media delivery techniques are defined ranging from centralized to decentralized ones. The older architectures are characterized by the presence of a single provider for multiple viewers. For this reason, it requires big amounts of resources at the server side. Broadcasting and multicasting are typical examples of this architecture. The newer architectures are capable of having multiple providers delivering media to multiple viewers. This

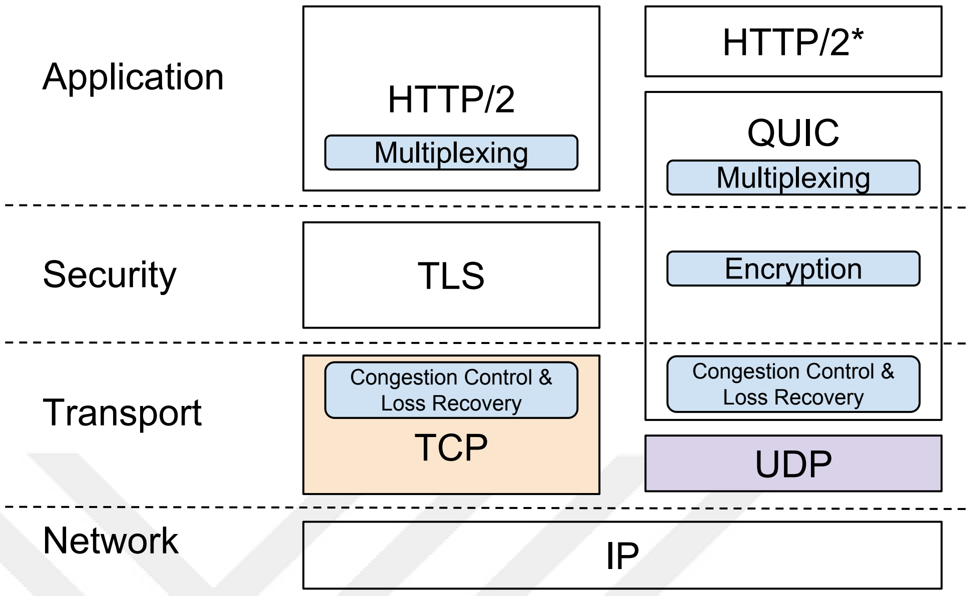

Figure 1: QUIC architecture (*: HTTP/2 shim).

architectures share and distribute the resource usage among multiple servers. Typical examples of this technique are peer-to-peer applications or video conferencing applications.

In on-demand services, media is delivered when requested, thus network resources are used only after a request received by the server. The server keeps a connection per client, which can handle the presentation and communicate with the server using another channel. In broadcast applications, media is delivered independently from requests, and is the same for all clients. In contrast to on-demand there is no connection. In this technique, clients join the broadcast, and the network resources are independent from the number of users. The major media delivery techniques are download and play, progressive download, traditional streaming and HTTP streaming. ***Download and Play*** is the simplest solution among all. In this technique, the client downloads the whole file, and then starts the playback of the content from its local disk. Nonetheless, a long initial delay is required before playback and it does not support live content. ***In progressive download***, multimedia content is downloaded progressively into a local buffer, from which the playback can start, as soon as sufficient data is available. However, it has some drawbacks. It wastes both network and local machine resources if the user stops watching content, and it does not provide any adaptation mechanism. ***Traditional streaming*** is a packet-based streaming requiring a state-full protocol, which sets up a session between the client and the server. ***HTTP streaming*** is file based approach that delivers appropriately structured files using HTTP. The deployment of HTTP streaming is simpler than traditional streaming, in fact it can take advantage of all the HTTP architecture in the existing Internet infrastructure.

In this thesis we primarily focus on HTTP streaming.

1.2 HTTP Streaming

HTTP streaming can be considered as an extension of progressive download, supporting both on-demand and live scenarios. It has become a very popular to deliver media content over the Internet. In fact, several commercial streaming platforms, such as Microsofts Smooth Streaming, Apples HTTP Live Streaming, and Adobes HTTP Dynamic Streaming, use HTTP streaming as delivery method. If a device wants to use a streaming service, it needs to support the specific protocol because each implementation defines its own manifest file (MPD) and segment format. The need for interoperability, required for rapid market growth, led to the development of MPEG DASH as a HTTP streaming standard.

1.3 MPEG DASH

Multimedia content consists of two parts in MPEG DASH streaming technology. First part is Media Presentation Description (MPD) file which gives information on the content properties such as representation alternatives and URLs of the segments. The second part includes segments containing the actual media content. First of all, MPD file should be downloaded from the server by the DASH client in order to play the multimedia content. Secondly, the client parses the MPD file for obtaining information on the media. By this process, the presentations, resolutions, media types and available bandwidths gathered by the client. Representation that best suits its needs can be selected and the streaming session can be started by requesting the media segments using HTTP with the help of this information. Subsequent segments can be downloaded and network bandwidth can be measured by the client during the streaming session. Segments which belong to different representations can be requested so that an enough buffer level can be kept. By this way adaptation into network changes can be done by the client. A high level overview of an MPEG DASH streaming session can be seen in Figure 2. The server containing media content at different bitrate levels can be seen on the left side; the representation of the network with variations of the available bandwidth over time can be seen in the middle and a client which downloads segments in different quality levels in response to the network bandwidth changes can be seen on the right side. The MPD file and the media segment formats can be defined by the scope of the MPEG DASH specification. However, the client behaviour for downloading segments, the media encoding formats along with the adaptation logic is outside of the MPEG DASH specification scope. An example of an MPEG DASH streaming scenario can be seen in Figure 3. In the Figure 3, the server contains the MPD and all the related segments can be seen on the left side. The MPD parser, the HAS adaptation algorithm, the HTTP library and the media player which are included by the DASH client application can be seen on the right side. The DASH client application contains the HAS adaptation algorithm which enables to manage the streaming session. It is responsible for the selection of the segments to be downloaded, using the information about the available bandwidth and the media presentation description in the MPD file. The HTTP library module sends the HTTP requests to the server, in order to download the selected segment, and provides the media to the media player. The media player is responsible for

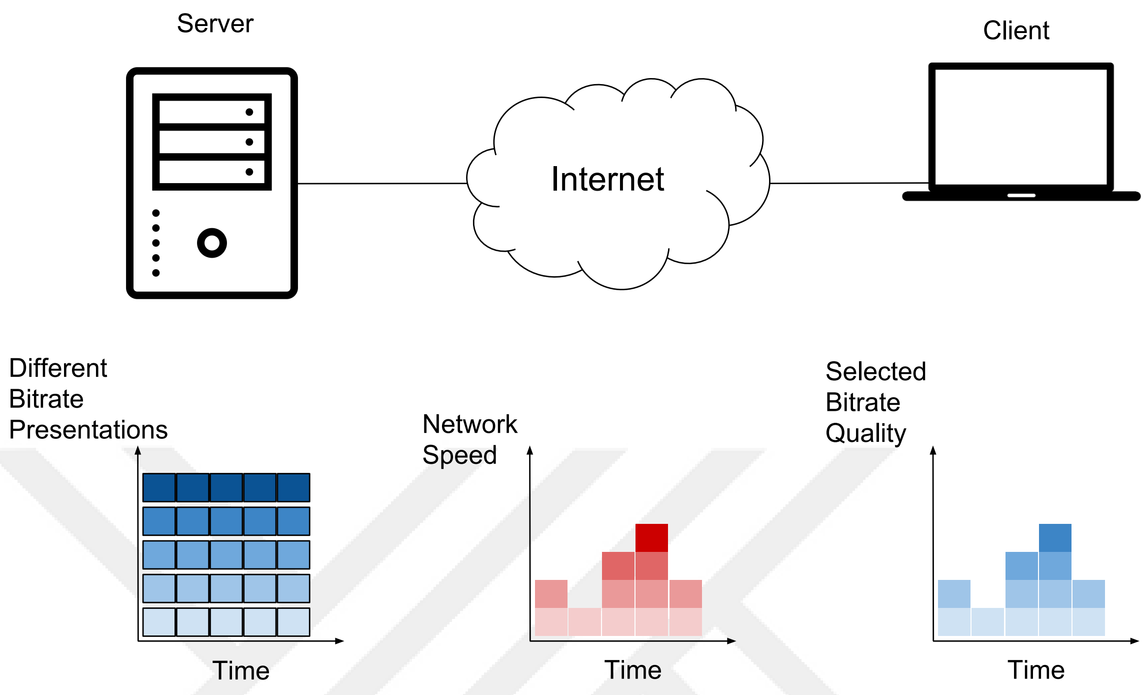

Figure 2: HAS player adaptation.

decoding and rendering of the media content.

The main MPEG DASH benefits are;

- Avoidance of problems with NAT traversal and firewalls,
- Reuse of the existing Internet infrastructure,
- Support for both on-demand and live streaming applications,
- Complexity is at the client side (HAS adaptation algorithm),
- Deployment of Content Delivery Networks (CDNs) can be kept simple,
- Seamless transition between different bitrate qualities,
- Support for different segment durations.

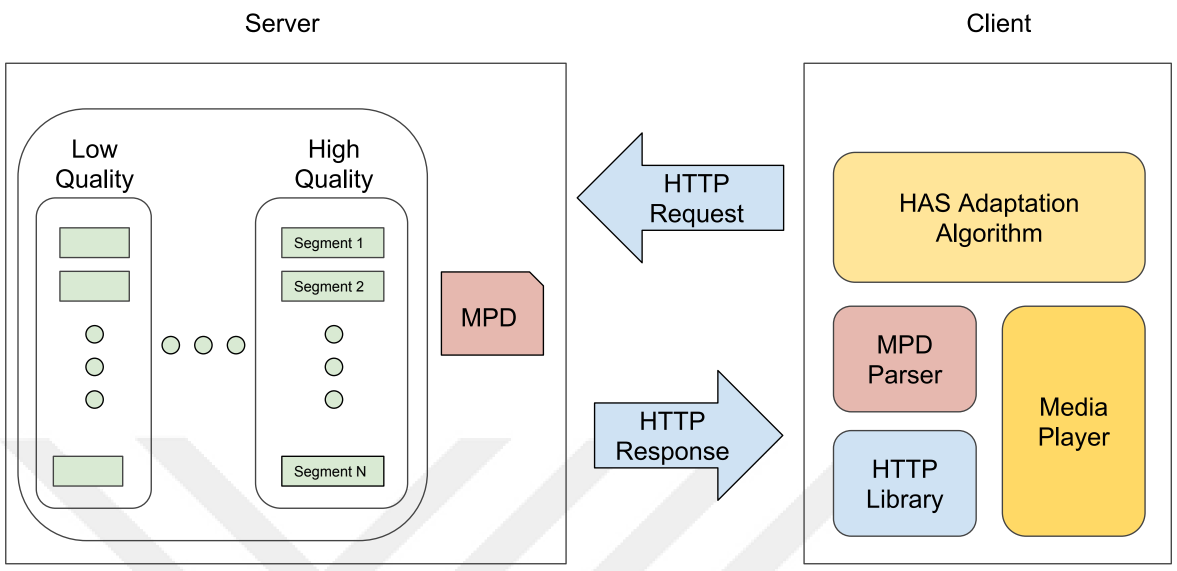

Figure 3: HTTP streaming architecture.

1.4 Head-of-Line Blocking in HAS Applications

The common belief that HTTP must 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 using TCP is mandatory. Rather, HTTP must used with a reliable transport protocol. In case of a packet loss, TCP blocks the subsequent packets that may have already been received in the receive buffer. In order to guarantee in-order delivery, TCP does not deliver received packets to the application layer unless the missing packet is recovered through a retransmission. 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 by the help of its superior design. 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. While TCP requires one and a half round-trip time (RTT) to complete the handshake, QUIC needs roughly half an RTT before the request is received by the server. 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.

1.5 Rebuffering Events in HAS Applications

The studies previously showed that viewers are quite sensitive to rebuffering events as even one percent increase in rebuffer rate can trigger a decrease in watch time more than three minutes [3]. Another research [4] showed that that the viewer engagement linearly decreased with increasing rate of rebuffer rate up to a certain threshold. Viewers watch only 30% of the video when there are more than 0.3 buffering events per minute. Experiencing more buffering events, the viewers got bored and 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. Moreover, 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 TCP connection.

1.6 Thesis Objectives

This thesis aims to answer the following three questions: First, on which conditions and how can QUIC start media streams more quickly, help to reduce the initial startup and seeking latency? Second, how can QUIC be beneficial to cope better with frequent connection changes and get better experience when the viewer is mobile and switches between the wireless networks? Third, 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? The HAS performance over QUIC is evaluated vs. over TCP on the Internet to answer the first two questions. A testbed environment is used to answer the third question.

This study is structured as follows: Chapter 2 presents an overview of other relevant works that studied streaming over QUIC. Chapter 3 describes the approach and setup that is designed to compare the performance of streaming clients running QUIC and TCP under different frame-seek scenarios and various network interface change scenarios. This chapter describes the approach for evaluating QUIC's multiplexing feature in a controlled network as well. Chapter 4 includes an analysis of the observed QoE measurements. Finally, Chapter 5 includes the conclusions, and a discussion of potential benefits of QUIC transport for HAS along with the future work.

CHAPTER II

RELEATED 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 [5]. 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 [6]. 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 [7]. 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 [8]. 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 [9]. According to Google's reports, QUIC reduced the rebuffer rates of YouTube by 18% and %15 for desktop users and mobile users, respectively [1]. Kakhki *et al.* reported that QUIC provided better streaming QoE than TCP, but only for high-definition video. 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 [10].

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 not random losses and QUIC performed better than SPDY (the pre-standard version of HTTP/2) when there were losses in the network [11]. 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 [12]. Cook *et al.* founded out that QUIC outperformed HTTP/2 over TCP/TLS in wireless mobile networks [13]. Qian *et al.* worked Web content delivery on mobile networks in a recent study. The authors found that multiple QUIC connections could cope with the restrictions of different congestion control algorithms when downloading short-lived Web content [14].

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 [15]. Furthermore, there was no markable increase for adaptive streaming performance with QUIC [8, 5]. We suspect that the conflicting results are primarily due to the fact that these studies tested either an older version of Google's QUIC server or an open-source (experimental) implementation that was not provided by Google. The QUIC's specification is being rapidly developed and deployed at users. For that reason, an older version of the code or a non-Google 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" [16]. However, in the evaluations for this thesis, we observed that the performance was good enough for testing with one client.

The differences compared to previous studies are listed below and summarized in Table 1.

- The latest public version of Google's QUIC implementation was used.
- Frame-seek scenarios were tested to investigate how much faster QUIC was compared to TCP.
- Scenarios involving wireless network disconnections/reconnections were tested.

- Experimentation with live video content.
- Focus on the wireless environments.
- Evaluation of multiplexing techniques for HAS applications in networks with packet loss and delay.

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

	Tested QUIC Version	Used Google's Server?	Wireless Networks[1]	Tested Different Algo-rithms?	Evaluated Frame Seeking?	Evaluated Conn. Switches?	Tested Live Video?
Timmerer [5]	v19	✓	✗	✗	✗	✗	✗
Szabo [6]	Latest [2]	✗	Only WiFi	✗	✗	✗	✗
Li [7]	Latest [3]	✓	✗	✗	✗	✗	✗
Bhat [8]	Latest [2]	✗	Only WiFi	✓	✗	✗	✗
Zinner [9]	Latest [3]	✓	✗	✗	✗	✗	✗
Kakhki [17	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.

CHAPTER III

APPROACH AND ENVIRONMENT SETUP

We tested frame-seek and connection switch scenarios on the public Internet (uncontrolled environment). To evaluate different multiplexing techniques, we used a testbed (controlled environment).

3.1 Internet Setup

In this setup, we did not use any test bed or any traffic shaping tool to throttle the network speed, produce loss or delay. We used the QUIC server (v39) provided by Google [16] for QUIC and Apache's HTTP server (v2.4.33) [18] 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 2 and Figure 4 show the network characteristics observed during the tests. The lower, middle and upper bars in the box plots in Figure 4 represent the 25th, 50th and 75th 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 4.

We made a couple of modifications in the Python-based player [19] 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 using a single QUIC connection over UDP. Second, the original player [19] used Python's *urllib* [20] to

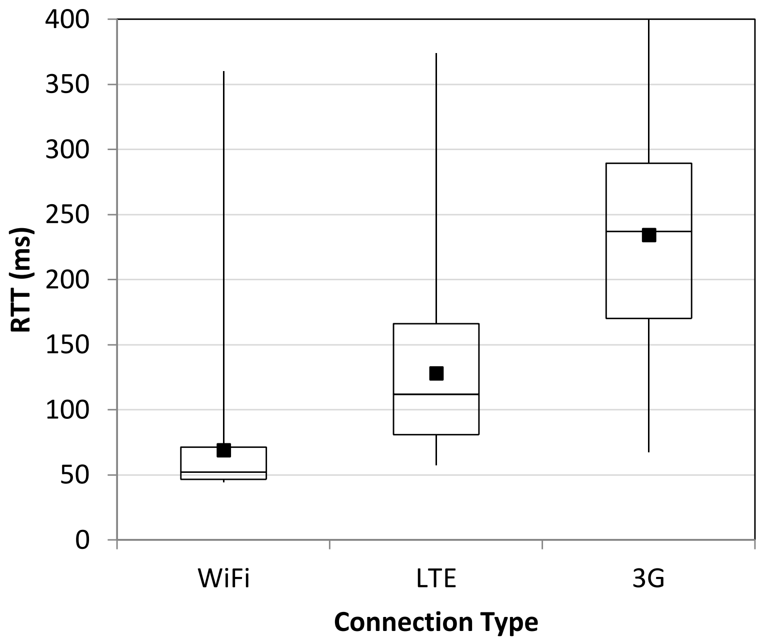

Figure 4: Measured RTTs during the tests (milliseconds).

Table 2: Measured network parameters (averages).

Type	Advertised Bandwidth	Measured Tput btw. Server & Client	Average RTT	Loss Rate
WiFi	50 Mbps	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%

make HTTP requests and it was creating a new connection for each media segment. However, QUIC transfers the data over a single UDP connection by default. A new TCP client was integrated instead of Python's *urllib* to download the files over a single TCP connection in order to make a fair comparison. The TCP client was coded using *libcurl* [21] and worked with HTTP keep-alive feature on. We also enabled Apache HTTP server's *KeepAlive* feature. Figure 5 presents how the HAS player works with TCP and QUIC. Both QUIC and TCP clients were coded in C++, and we made some changes in Google's QUIC server code to disable its in-memory cache. The modified player code, modified QUIC server code, the QUIC and TCP clients are available for public access on GitHub for the research community [22].

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 [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 determine the time required to download the next segment. This algorithm predicts the throughput with a weighted harmonic mean method. [19].

- **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 [23].

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.

3.1.1 Frame-Seek Scenario

This section describes our test environment setup, modifications that we made in the player and 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 scripted 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 3 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.

3.1.2 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,

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

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 6 shows the network setup for these evaluations and the connection switch scenario is shown in Table 4.

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

3.2 Testbed Setup

3.2.1 Evaluating QUIC's Multiplexing Feature

In this section, we focus on evaluating QUIC's multiplexing features in a controlled environment. We used a testbed to evaluate different multiplexing techniques. 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 7 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 7 using the Network Emulator (tc-NetEm) tool [24]. 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 also emulated the respective RTT and packet loss rate.

We used two different throughput profiles to throttle the bandwidth in the emulated tests. The FCC profile [25] provides a profile with a bandwidth variation of 0.8 Mbps to 1.4 Mbps while the DASH-IF NP2b profile [26] 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 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 [26]. 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. For the typical network emulation, we used the following set of RTTs and packet loss rates sequentially: {(100 ms, 0.12%), (88 ms, 0.09%), (75 ms, 0.06%), (50 ms, 0.08%), (38 ms, 0.09%), (50 ms, 0.08%), (75 ms, 0.06%), (88 ms, 0.09%)}. For the large delay emulation scenario, we used the following RTTs: {200 ms, 400 ms, 600 ms, 800 ms, 1300 ms, 800 ms, 600 ms, 400 ms, 200 ms}. For the large loss emulation, we used the following packet loss rates: {0.1%, 1%, 2%, 5%, 10%, 5%, 2%, 1%, 0.1%}. All values were taken from the DASH-IF guidelines [26]. We used an on-demand content from a dataset [27] in these tests. The segment duration and playback buffer size were four and 20 seconds, respectively.

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 [28]. 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 head-of-line blocking if it is used over TCP. In HTTP/2 technique, 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 8:

(a) HTTP/1.1 over multiple TCP connections,

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

(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) [29]. In the HTTP/2 approach, we used *libcurl* [21] HTTP/2 implementation with multiplexing and pipelining 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 [1]. In our experiments, we did not change this setting. The maximum packet size for QUIC was 1392 bytes (including the headers) in our tests.

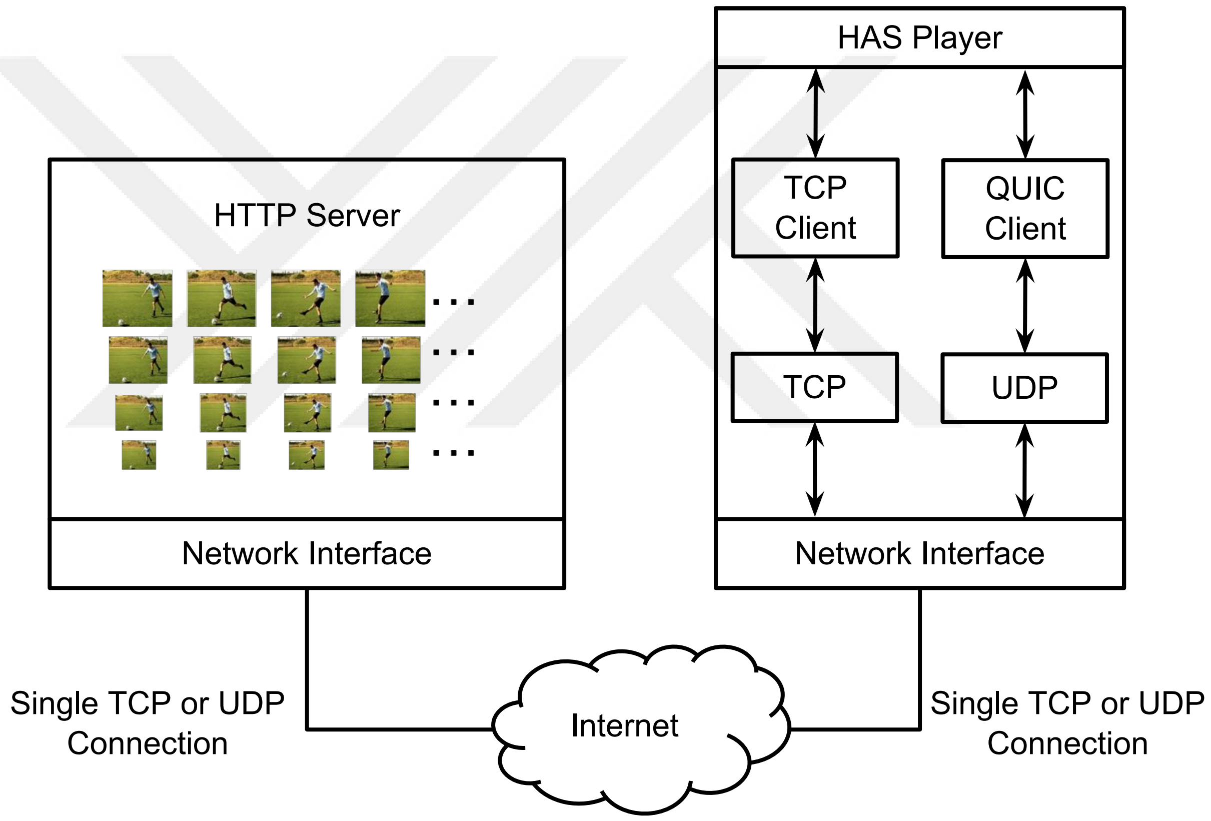

Figure 5: The HAS player can use QUIC or TCP to download the media segments.

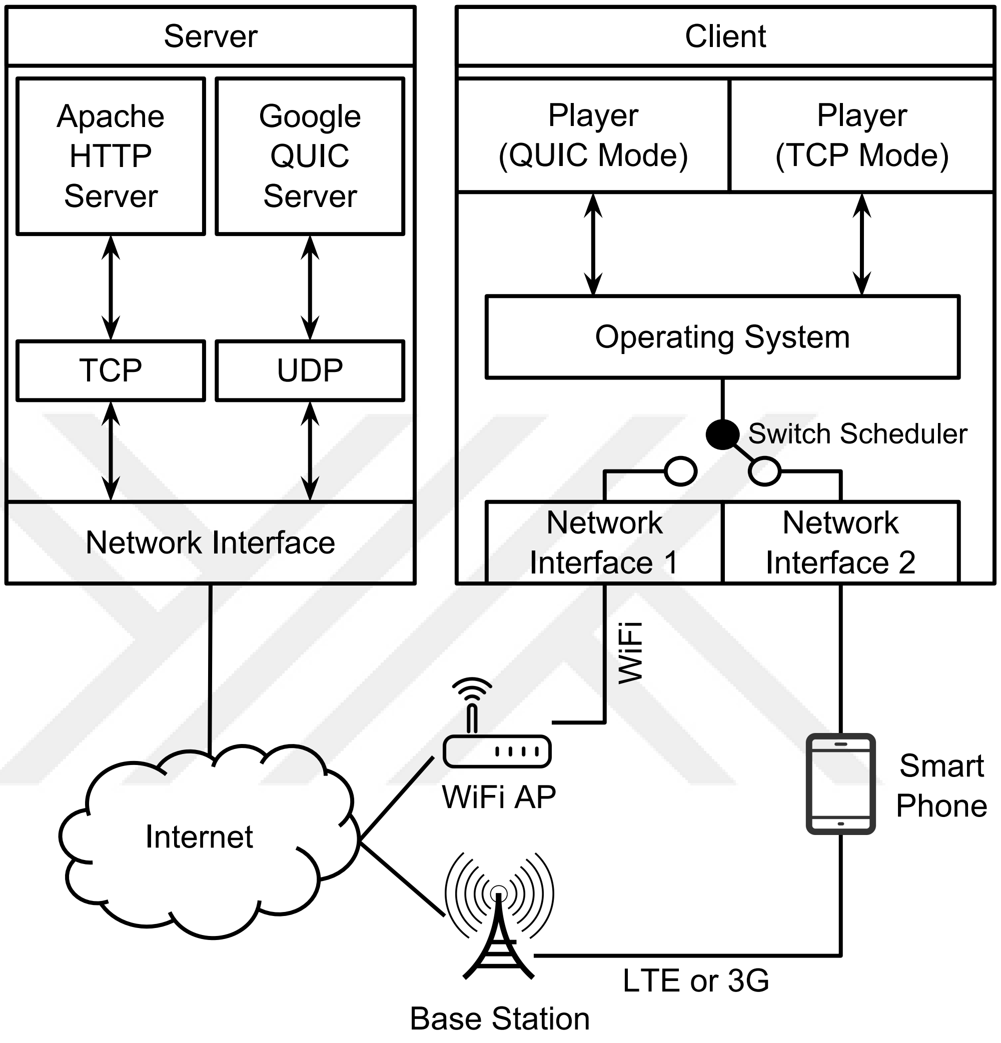

Figure 6: Internet setup for the connection-switch tests.

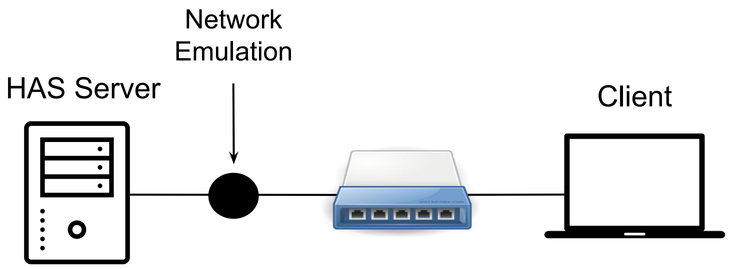

Figure 7: Testbed setup for controlled experiments.

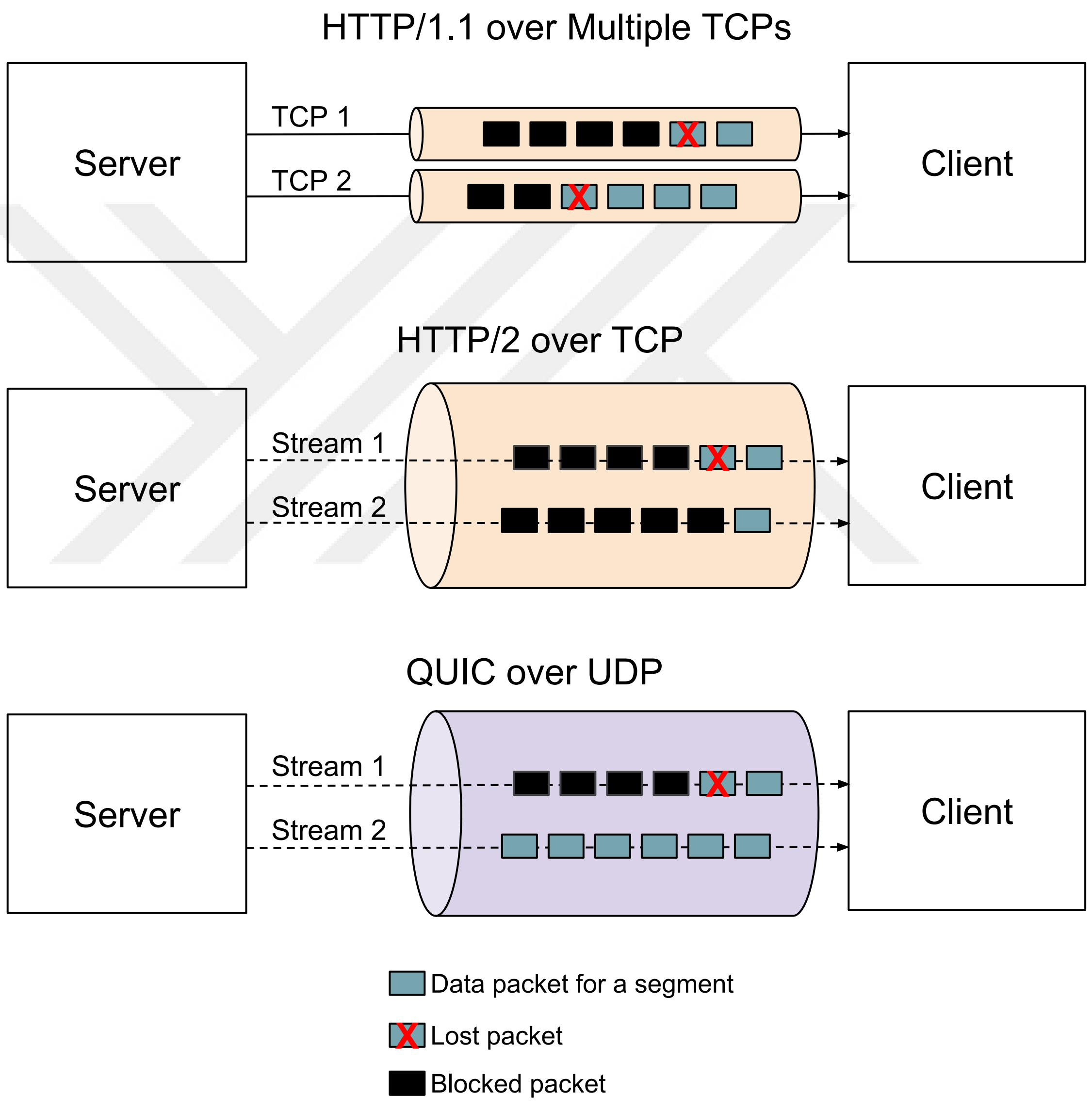

Figure 8: Sending multiple streams of data.

CHAPTER IV

RESULTS

For the uncontrolled environment, 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 the 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. Furthermore, each test was repeated 10 times to prevent substantial anomalies. In this section, we present the average results.

In our evaluations, we used 600-second long content with 20 different bitrates [27]. The bitrates ranged from 44 Kbps to 3.9 Mbps. 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 Measured Metrics

- *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 [30].

- *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 [31, 32]. We measured this metric only for the frame-seek scenarios.

- *Rebuffer Rate*: The rebuffer rate is calculated as follows:

$$\text{Rebuffer Rate} = \frac{\text{Rebuffer Time}}{\text{Rebuffer Time} + \text{Media Play Time}} \tag{1}$$

  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 in the uncontrolled environment, 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.

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 performed a shorter average wait time after seeking and started the media streams more quickly. QUIC achieved reducing the wait times by up to 50% percent compared to TCP. QUIC reduced the rebuffer rates as well. Our results may seem to conflict with some of the previous research [8]. We suspect that the contradictory nature of [8] compared to ours is primarily due to the fact that an open-source server implementation with experimental QUIC support [33, 34] 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.

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%

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 a half seconds. This is an important result, considering that keeping the average seek time less than two seconds is essential for viewer engagement and loyalty [31, 32].

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 RTT than the WiFi network as shown in Table 2. Google reported that QUIC's benefits were higher when congestion and delays were higher in the network [1]. Our observations confirm this with the results for the 3G network. However, our result conflicts with [17] 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.

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%

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 6 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 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

[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%

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 window more aggressively than TCP and is able to achieve a larger congestion window when competing with TCP [17]. Hence, we can say 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 provided 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 Feature

During the tests, the available bandwidth for all types of multiplexing techniques were high enough to stream without causing a stall or rebuffer event except for the

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%

large delay and large loss scenario.

- 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 for HTTP/1.1's disadvantage. With eight or more connections, the average playback bitrate started to decrease for HTTP/1.1. QUIC performed similar to HTTP/1.1 over two connections in this scenario and performed worse than HTTP/1.1 over more than two connections. However, QUIC provided a higher average playback bitrate than HTTP/2 in all cases.

- 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.

- For the large delay and large loss scenario, QUIC performed better than HTTP/1.1 over two or more connections in terms of rebuffer rates and performed slightly worse than HTTP/1.1 in terms of average playback bitrate. HTTP/1.1 over one TCP connection showed the lowest rebuffer rate. The HTTP/2 approach performed worse than the others when there is large loss or large delay in the network. For HTTP/2, the rebuffer rate increased slightly as the number of parallel requests increased.

The results are shown in Table 10.

Table 10: Results are averaged for BASIC, SARA and BBA-2 algorithms as well as DASH-IF NP2b and FCC network profiles.

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

CHAPTER V

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 enables viewers to have a seamless transition among different networks. Even though the CID was not used in our tests, we saw that QUIC outperformed TCP in terms of average playback bitrate and rebuffer rate.

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 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 [35]. 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.

Finally, we 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.

Bibliography

[1] 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," Proceedings of the Conference of the ACM Special Interest Group on Data Communication, 2017.

[2] Sandvine, "Global Internet Phenomena Report," 2016.

[3] 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.*, vol. 41, no. 4, pp. 362–373, 2011.

[4] 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.*, vol. 43, no. 4, pp. 339–350, 2013.

[5] C. Timmerer and A. Bertoni, "Advanced transport options for the dynamic adaptive streaming over HTTP," *Computing Research Repository*, vol. abs/1606.00264, 2016.

[6] G. Szabo, S. Racz, D. Bezzera, I. Nogueira, and D. Sadok, "Media QoE enhancement with QUIC," 2016 IEEE Conference on Computer Communications (INFOCOM) Workshops, 2016.

[7] B. Li, C. Wang, Y. Xu, and Z. Ma, "An MMT based heterogeneous multimedia system using QUIC," 2016 2nd International Conference on Cloud Computing and Internet of Things (CCIOT), 2016.

[8] D. Bhat, A. Rizk, and M. Zink, "Not so QUIC: a performance study of DASH over QUIC," Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video, 2017.

[9] T. Zinner, S. Geissler, F. Helmschrott, and V. Burger, "Comparison of the initial delay for video playout start for different HTTP-based transport protocols," 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2017.

[10] I. Ayad, Y. Im, E. Keller, and S. Ha, "A practical evaluation of rate adaptation algorithms in HTTP-based adaptive streaming," *Computer Networks*, vol. 133, pp. 90 – 103, 2018.

[11] G. Carlucci, L. De Cicco, and S. Mascolo, "HTTP over UDP: an experimental investigation of QUIC," Proceedings of the 30th Annual ACM Symposium on Applied Computing, 2015.

[12] P. Megyesi, Z. Kramer, and S. Molnar, "How quick is QUIC?," 2016 IEEE International Conference on Communications (ICC), 2016.

[13] S. Cook, B. Mathieu, P. Truong, and I. Hamchaoui, "QUIC: Better for what and for whom?," 2017 IEEE International Conference on Communications (ICC), 2017.

[14] P. Qian, N. Wang, and R. Tafazolli, "Achieving robust mobile Web content delivery performance based on multiple coordinated QUIC connections," *IEEE Access*, vol. 6, pp. 11313–11328, 2018.

[15] Y. Yu, M. Xu, and Y. Yang, "When QUIC meets TCP: an experimental study," 2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC), 2017.

[16] "Playing with QUIC." `http://www.chromium.org/quic/playing-with-quic`, accessed Sep. 2017.

[17] 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," Proceedings of the 2017 Internet Measurement Conference, 2017.

[18] "Apache HTTP server." `http://httpd.apache.org`, accessed Jan. 2018.

[19] P. Juluri, V. Tamarapalli, and D. Medhi, "SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP," 2015 IEEE International Conference on Communication Workshop (ICCW), 2015.

[20] "Python urllib." `http://docs.python.org/2/library/urllib.html`, accessed Dec. 2017.

[21] "libcurl." `http://curl.haxx.se/libcurl`, accessed Oct. 2017.

[22] "GitHub quic-streaming." `http://github.com/sevketarisu/quic-streaming`, accessed March. 2018.

[23] 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," Proceedings of the 2014 ACM Conference on SIGCOMM, 2014.

[24] "tc-netem." `http://wiki.linuxfoundation.org/networking/netem`, accessed Jan. 2018.

[25] "F. c. commission. raw data - measuring broadband america.." `https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016`, accessed June. 2018.

[26] "Dash-if guidelines." `https://dashif.org/wp-content/uploads/2016/06/DASH-AVC-264-Test-Vectors-v1.0.pdf`, accessed June. 2018.

[27] "DASH dataset." `http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014`, accessed Sept. 2017.

[28] "Push technology." `http://docs.pushtechnology.com/cloud/latest/manual/html/designguide/solution/support/connection_limitations.html`, accessed June. 2018.

[29] R. Fielding and J. Reschke., "Rfc 7230: Hypertext transfer protocol (http/1.1): Message syntax and routing," 2014.

[30] 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*, vol. 19, no. 10, pp. 2136–2151, 2017.

[31] Akamai, "Maximizing audience engagement: how online video performance impacts viewer behavior," 2015.

[32] Conviva, "OTT streaming market year in review," 2017.

[33] "Caddy QUIC support." `http://github.com/mholt/caddy/wiki/QUIC`, accessed Jan. 2018.

[34] "quic-go issues." `http://github.com/lucas-clemente/quic-go/issues/302`, accessed Jan. 2018.

[35] A. C. Begen and Y. Syed, "Are the streaming format wars over?," IEEE Int. Conf. Multimedia and Expo (ICME), July 2018.

VITA

Şevket Arısu was born in Isparta, Turkey. He received his B.Sc. in Computer Science from Istanbul University in 2006. He is currently working as a senior software engineer at Turkcell. Mr. Arısu started to study Master of Science Program in Computer Science Department of Özyeğin University in 2015. His research interests include HTTP adaptive streaming, QoE, QUIC and transport options for media.