Improving QoE of ABR Streaming Sessions through QUIC Retransmissions

Divyashri Bhat, Rajvardhan Deshmukh, Michael Zink
University of Massachusetts Amherst
dbhat@ecs.umass.edu,rdeshmukh@umass.edu,zink@ecs.umass.edu

ABSTRACT

While adaptive bitrate (ABR) streaming has contributed significantly to the reduction of video playout stalling, ABR clients continue to suffer from the variation of bit rate qualities over the duration of a streaming session. Similar to stalling, these variations in bit rate quality have a negative impact on the users' Quality of Experience (QoE). In this paper, we use a trace from a large-scale CDN to show that such quality changes occur in a significant amount of streaming sessions and investigate an ABR video segment retransmission approach to reduce the number of such quality changes. As the new HTTP/2 standard is becoming increasingly popular, we also see an increase in the usage of QUIC as an alternative protocol for the transmission of web traffic including video streaming. Using various network conditions, we conduct a systematic comparison of existing transport layer approaches for HTTP/2 that is best suited for ABR segment retransmissions. Since it is well known that both protocols provide a series of improvements over HTTP/1.1, we perform experiments both in controlled environments and over transcontinental links in the Internet and find that these benefits also "trickle up" into the application layer when it comes to ABR video streaming where QUIC retransmissions can *significantly* improve the average quality bitrate while simultaneously *minimizing* bit rate variations over the duration of a streaming session.

KEYWORDS

ABR streaming, DASH, QUIC, HTTP/2, QoE

ACM Reference Format:
Divyashri Bhat, Rajvardhan Deshmukh, Michael Zink . 2018. Improving QoE of ABR Streaming Sessions through QUIC Retransmissions. In *Proceedings of ACM Conference (MM'18).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

After two decades, the HyperText Transfer Protocol has undergone a significant makeover resulting in the introduction of the HTTP/2 standard [26] gaining notable popularity in the Internet, where it is currently used by 24.6% of all web sites [4]. HTTP/2 makes several improvements over its predecessor HTTP/1.1 [10]. These improvements include *a)* multiplexing, where streams for multiple requests can be sent over a single TCP session; *b)* header compression; and, *c)* an option where the web server can push content to the client proactively. HTTP/2 has been specified to use TCP as the underlying transport protocol. This combination of HTTP/2 and TCP has several performance issues, including a delay introduced by the 3-way handshake for each connection setup (this is even higher if Transport Layer Security (TLS) is used). In addition, the issue of head of line (HOL) blocking still exists. The Quick UDP Internet Connections protocol (QUIC) [20] is a new approach designed to combine the speed of UDP with the reliability of TCP and, thus overcome these issues. QUIC has been specifically designed to reduce latency of web page loads and mitigate rebuffers in video streaming clients.

Adaptive bitrate (ABR) streaming has become the de-facto streaming standard for video on demand platforms such as Netflix [1] and Youtube [5]. With more than 70% of the peak hour US Internet traffic [30], video streaming has become *the* killer-application of today's Internet. ABR video streaming solutions like Dynamic Adaptive Streaming over HTTP (DASH) [25] are, however, stuck in an HTTP/TCP setting that has been shown to possess substantial drawbacks with respect to Quality-of-Experience (QoE) [15, 33].

Recently, we proposed a DASH-based ABR approach (SQUAD) [33] that has the goal to improve QoE for viewers watching video streams over the Internet. One specific feature of SQUAD is the ability to retransmit segments[1] in a higher quality than they were originally transmitted in [32] to reduce frequent quality changes during a streaming session. The drawback of implementing this approach on top of HTTP/1.1 is the inability to efficiently schedule such retransmissions. In the case of one TCP session, retransmission requests[2] have to be interleaved with requests for new original segments, that have not been requested in the past. Parallel transmissions require the setup of a new connection, which comes with the drawback of additional delay due to the 3-way handshake. The use of HTTP/2 over TCP makes such retransmissions more efficient, since they can be scheduled within the same TCP connection. While HTTP/2 has the potential to improve the performance of SQUAD in the case of retransmission, the impact of losses and the resulting HOL blocking has not been studied. In addition, it has not been evaluated to what extent QUIC can further improve SQUAD with retransmissions, since it eliminates the HOL blocking issue.

In this paper, we make the following contributions:

- An analysis of 5 million video streaming sessions reveals that switches in quality representations that result in a gap

MM'18, October 2018, Seoul, Korea

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] In the remainder of this paper, we use the word segments to denote ABR video segments unless specified otherwise.
[2] In the remainder of this paper, we use the word retransmissions to denote retransmissions of a received video segment in a higher quality unless specified otherwise.

occur in almost 36% of all streaming sessions. In the case of mobile clients this number increases to 50%.
- We make a systematic comparison of the multiplexing feature of HTTP/2 and QUIC, particularly for retransmitting ABR video segments in a higher quality with the objective of improving the overall QoE of ABR streaming sessions.
- Our evaluation results show that QUIC retransmissions can *significantly* improve the average quality bitrate while simultaneously *minimizing* bit rate variations over the duration of a streaming session.

The remainder of this paper is structured as follows. In Sect. 2, we present background information on HTTP/2 and QUIC, as well as retransmission scheduling, and an analysis of more then 5 million streaming sessions from a CDN. Sect. 3 details how segment retransmission is performed in the case of HTTP1.1, HTTP/2, and QUIC and an evaluation of this approach using controlled and Internet measurements is presented in Sect. 4. Related work is presented in Sect. 5 and Sect. 6 concludes the paper.

2 BACKGROUND

In this section, we give an overview of the differences between HTTP/2 over TCP and QUIC and introduce the QoE metrics we use to evaluate the performance of a variety of ABR streaming approaches. This is followed by an overview of our retransmission scheduling approach described in [32].

2.1 QUIC vs. TCP

With the advent of QUIC there are now two options for the transport of HTTP/2 sessions between a web server and a browser. First, the original approach specifies the use of HTTP/2 over TCP. The second approach involves QUIC as an additional application layer protocol, which results in a HTTP/2 over QUIC over UDP solution [3].

TCP requires a 3-way handshake resulting in a 1.5 RTT before any data request is received at the server. In the case of QUIC, the data request arrives after 0.5 RTT at the server.

The default congestion control algorithm implemented in QUIC is similar to that of TCP Cubic [13] with some important differences. In order to notify the sender of the train of packets received, existing TCP mechanisms (including CUBIC) make use of Selective Acknowledgements (SACK) that include a maximum of the 3 most recent sequential packets that arrived successfully. The sender then retransmits the lost packets with sequence numbers that lie within the range of the 3 SACKs received. It is obvious that this approach imposes a heavy constraint on the number of TCP retransmissions that can take place without response from the receiver. QUIC aims to resolve this by including the use of NACKs and allows the receiver to send up to 256 NACKs without waiting for a response from the sender. The use of NACKs allows much faster loss recovery and can lead to asignificant reduction in rebuffering at DASH clients.

2.2 QoE metrics for ABR Streaming

For the evaluation of the performance of ABR streaming applications we make use of the following metrics, which are widely used in related work:

Average Quality Bitrate (*AQB*): One of the objectives of quality adaptation algorithms is to maximize the average quality bitrate of the streamed video. For a comprehensive QoE representation, we need to combine this metric with the *Number of Quality Switches* which is explained below.

Number of Quality Switches (#*QS*): This metric is used together with *AQB* to draw quantitative conclusions about the perceived quality (QoE). For example, for two streaming sessions having the same *AQB*, the session with the lower #*QS* will be perceived better by the viewer [35].

Spectrum (*H*) [35]: The spectrum of a streamed video is a centralized measure for the variation of the video quality bitrate around the *AQB*. A lower *H* indicates a better QoE.

Rebuffering Ratio (*RB*): The average rebuffering ratio is given by the following equation:

$$RB = \mathsf{E}\left[\frac{t_a - t_e}{t_e}\right], \tag{1}$$

Where t_a is the actual playback time and t_e is the video length in seconds, respectively.

It is well known that low rebuffering and high average quality bitrate are highly desirable for an optimal QoE. In our previous work [32], we show that reducing quality gaps through ABR segment retransmissions can contribute significantly to a higher *AQB*. In the following, we present an analysis of actual quality gaps that occur in a real-world trace.

2.3 Segment retransmission scheduling

Traditional ABR approaches stream the ABR video segments in the order provided by the MPD file. Looking closely at the segment qualities buffered at the client at any point in time SQUAD shows that these reflect the recent quality decisions made by the adaptation algorithm, which, in turn, are based on the specific interpretation of the measured download rate and the corresponding buffer filling. Looking at the buffer filling in *retrospect* as in Fig. 1 (a) SQUAD identifies quality switches that are denoted as quality "gaps". The emergence of these quality gaps is complex as it describes the instantaneous interaction of the adaptation algorithm with the buffer filling state and the download rate. In the following, we illustrate how to improve the QoE by filling some of these quality gaps. Fig. 1 shows a simplified example of segment qualities inside the player buffer with different possible gaps. SQUAD defines gaps as the downward variation from the quality level which negatively impact the QoE [35].

SQUAD's current approach is based on HTTP/1.1, which does not allow the parallel transmission of original segments and the retransmission of segments in a better quality. HTTP/2 over TCP allows this parallel transmission but does not prevent HOL blocking to efficiently perform retransmissions. In contrast, HTTP/2 over QUIC does not suffer from such inefficiencies and gives the application maximum control of individual streams and we show how this can be used to improve the QoE of ABR streaming.

2.4 Analysis of Gaps in Streaming Sessions

Akamai [23] is the world's largest CDN provider that delivers 15%–30% of global Internet traffic. Its CDN contains over 150,000 edge servers distributed in 90+ countries and 1200 ISPs around the world. To motivate the retransmission of segments as described in Sect. 2.3,

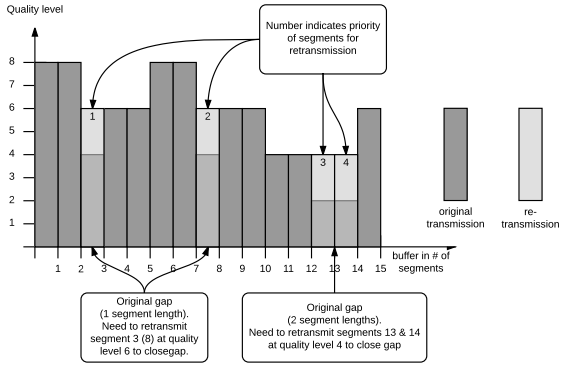


Figure 1: Example scenario for retransmissions. The QoE of this streaming session can be improved if, e.g., segments 3, 8, 13, and 14 are retransmitted in higher quality, assuming they arrive before their scheduled playout.

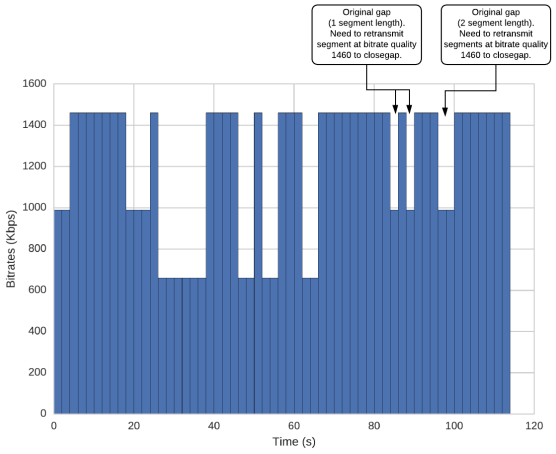


Figure 2: Original transmission of video stream from one randomly selected trace in the Akamai data set. The QoE of this video can be improved if the highlighted segments are retransmitted in higher quality, assuming they arrive before scheduled playout.

we analyze an anonymized trace collected from Akamai's video CDN. This trace contains video streaming session information for a 3-day period in June 2014. The ABR streaming traffic in this trace contains 5 million video sessions originating from over 200,000 unique clients who were served by 1294 edge servers around the world. For each streaming session, each individual segment request is logged, which allows us to reconstruct the quality of the segments received at the client. Fig. 2 gives an example for one such streaming session we randomly picked for better illustration. As shown in Fig. 2, this streaming session resulted in a series of gaps. These gaps are potential candidates for segment retransmission that could lead to less quality level changes and, thus, an improve QoE. In Fig. 2, we indicate that a retransmission of the segments in the later part of the stream could significantly impact the QoE.

In Fig. 3, we show the results of our analysis for the complete data set which has approx. 5 million sessions, and for the a subset

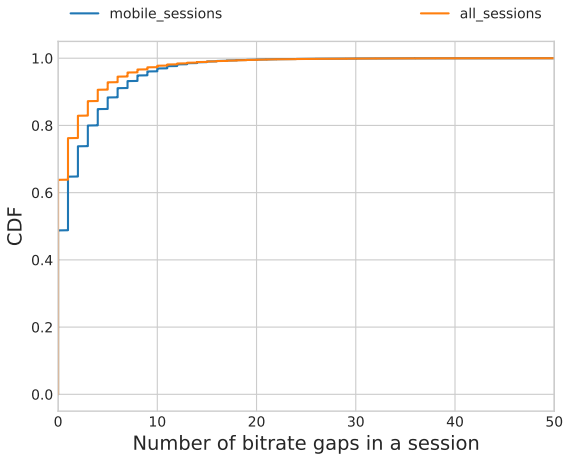


Figure 3: CDF for the number of sessions with one or more gaps for all sessions (orange) and mobile sessions (blue)

that only includes sessions for mobile devices, which has approx. 0.1 million sessions. This figure shows the percentage of sessions that have one or more gaps. Considering all sessions in the data set, 36.19% of the sessions have at least one gap. These sessions could benefit from our segment retransmission approach. We also analyzed how many of the sessions with mobile clients have at least one gap. As shown in Fig. 3, with 51.24% this ratio is even higher. Obviously, this increase in sessions with at least one gap is not too surprising since mobile/wireless clients are assumed to experience higher bandwidth fluctuations than stationary/wired clients.

3 SEGMENT RETRANSMISSON OVER HTTP/2

After introducing the basics of segment retransmission for ABR streaming and outlining its drawbacks in the case of HTTP/1.1 in Sect. 2.3, we introduce the usage of this approach in the case of HTTP/2 in this section. We first give an overview on the advantages and disadvantages of using our retransmission approach in the case of HTTP/2 over TCP and QUIC, respectively. This is followed by a description of the implementation of our approach. Results from an evaluation of this approach are presented in Sect. 4.

3.1 Example

In the following, we compare a segment retransmission approach that is based on HTTP/2 over TCP (the current standard) with one that is based on HTTP/2 over QUIC. The TCP-based approach is shown in Fig. 4. Here, we show a specific scenario of retransmissions for ABR streaming. In this scenario, HTTP/2 over TCP allows the multiplexing of multiple requests within a single TCP connection. This feature makes this approach more efficient than our existing HTTP/1.1 solution, since original segment transmissions and retransmission can be performed in parallel. (In the case of HTTP/1.1, segments can either only be transmitted sequentially or a new TCP connection has to be established for the retransmissions). Despite the support of multiplexing several requests over a single TCP connection, this approach has several drawbacks. First of all, HOL blocking can lead to stalling. Such a case is indicated

in Fig. 4, where the first retransmitted TCP segment is lost. All of the following (original and retransmitted) TCP segments will be blocked from delivery to the application layer until the lost TCP segment is successfully received. This HOL blocking, caused by a TCP segment retransmission, prevents original segments from being delivered to the buffer of the video player. This can result in an incorrect estimate of the segment download rate and consequently an unnecessary reduction in bit rate quality for the download of future original segments. In the worst case, this causes the drainage of the video player buffer, which in turn will stall the video playout.

In contrast, an HTTP/2 over QUIC approach is not impacted by HOL blocking. Fig. 5, shows the same segment transmission scenario as in Fig. 4. As opposed to the scenario shown in Fig. 4, the QUIC-based approach does not prevent the original datagrams from being delivered to the video player buffer if the first retransmitted UDP datagram is lost. This should lead to a significant reduction in the risk of stalling and misinterpretation of the download rate. In addition, the application can decide if the lost retransmitted UDP datagram should be retrieved again or not. This decision can be based on buffer fill level, position of the retransmitted segment in the buffer, and observed download rate. With the use of QUIC, the application can also determine at which rate segments should be downloaded. For example, pacing [2] can be applied for the retransmission of segments to assure that such transmissions only minimally interfere with the transmission of original segments.

3.2 Implementation

In this section, we give an overview of our implementation of ABR streaming that is based on HTTP/2 over QUIC that enables retransmissions of segments that have originally been transmitted in a low(er) quality (see Sect. 2.3).

3.2.1 SQUAD with HTTP/2 and HTTP/1.1. Since the multiplexing feature of HTTP/2 is unavailable in its predecessor, HTTP/1.1, the original version of SQUAD implements retransmission scheduling as a series of GET requests where at any given time there is only one outstanding request to the ABR streaming server. Intuitively, such a sequential implementation stalls the application pipeline and can lead to either conservative retransmission scheduling or a severe buffer drain. To prevent stalling in case of a severe drop in measured download rate, SQUAD implements retransmission abandonment, which cancels segment retransmission when we observe that the segment will not be downloaded on time. Our HTTP/2 implementation converts this sequential behavior into a parallel, multiplexed session of two simultaneous GET requests, where, at any given time there are a maximum of two possible streams active within a single connection. SQUAD is implemented as part of an open-source Python-based DASH player emulator, `AStream`.[3] For ease of integration, we use the Python-based HTTP/2 library, `hyper`[4], in order to implement two multiplexed GET requests for original and retransmission segment downloads. Additionally, we implement multithreading to allow transmissions on both HTTP/2 streams to proceed independently. We note that HTTP/2 still uses the same TCP connection which suffers from HOL blocking as explained in Sect. 3.1. and therefore, we also implement a SQUAD over QUIC approach which is introduced next. In order to make a fair comparison, we also adapt the original implementation of SQUAD to use `hyper` for making HTTP/1.1 requests.

3.2.2 SQUAD with QUIC. Similar to the experiment above, we implement multiplexed sessions for original and retransmitted segment downloads using QUIC. However, we include the use of IPC message streams with minimal overhead to communicate between the QUIC client (implemented in C++) and the AStream player (implemented in Python). Unlike HTTP/2 over TCP, QUIC does not suffer from HOL blocking and is designed to deliver data to the application as soon as they arrive at the receiver and a stream within a QUIC connection is not adversely affected by events that cause delay or loss of packets on a parallel, ongoing stream. At the time of this implementation, we used Chromium for Linux with QUIC version `Q043`. In order to provide support for multiplexed streams for SQUAD, we perform the following modifications on the `QUIC client`[5] code provided by Google: (i) we create and synchronize simultaneous streams within a single connection, (ii) we introduce IPC messaging not only to send commands between AStream and QUIC but also to provide intermediate chunk download rate measurements to the SQUAD ABR algorithm.

We note that this work does not focus on modifying SQUAD to perform optimally with a protocol such as QUIC but is instead intended as a study to evaluate the performance of SQUAD retransmissions over QUIC in order to determine if QoE can be improved with such an approach.

4 EVALUATION DESIGN

In this section, we describe a series of experiments, which are specifically designed to study the QoE performance of using QUIC and HTTP/2 for ABR video streaming with a focus on segment retransmission. We compare the results of these experiments with the baseline approach that uses HTTP/1.1. The server nodes (denoted as *Server1 - Server4* in Fig. 6) run a Caddy server (version=0.10.10) [6] with the experimental QUIC mode enabled such that the clients can stream DASH videos either over TCP or QUIC. We chose the Caddy server as it is a production server which is capable of simultaneously supporting QUIC, HTTP1.1, and HTTP/2 over TCP with TLS1.2. All experiments use an excerpt of the `BigBuckBunny` dataset [21] (unless stated otherwise) that comprises a 300s-long video with a 2s segment duration and the corresponding MPD file. We extended the MPD file by providing the size of each segment in each of the available quality levels.[6] The quality bitrates available in this MPD file are the following: {0.09, 0.13, 0.18, 0.22, 0.26, 0.33, 0.59, 0.79, 1.03, 1.24, 1.54, 2.48, 3.52, 4.21}Mbps. The client nodes run the SQUAD ABR algorithm [32] described above, which is implemented in a Python-based DASH player [18].

4.1 Testbed

For our controlled experiments, we use Cloudlab [24] which is a geographically distributed testbed for the development, deployment, and validation of cloud-based services. The CloudLab infrastructure

[3] https://github.com/pari685/AStream
[4] https://github.com/Lukasa/hyper
[5] https://www.chromium.org/quic/playing-with-quic
[6] We use segment sizes in the MPD file since this was introduced in AStreamer. This can easily be replaced by using byte ranges, which are available in real-world, ABR streaming solutions.

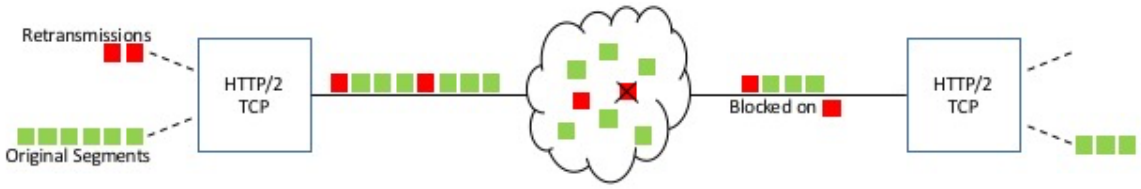


Figure 4: This figure shows a scenario of original and retransmitted segment transmission in the case of HTTP/2 over TCP. The first of the retransmitted TCP segments (red) is lost, which leads to HOL blocking at the receiver.

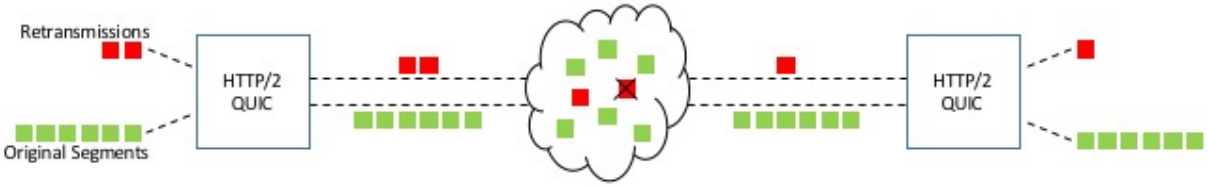


Figure 5: This figure shows a scenario of original and retransmitted segment transmission in the case of HTTP/2 over QUIC. In contrast to Fig. 4, the loss of a retransmitted UDP datagram (red) does not lead to HOL blocking and all original segments are delivered to the video player buffer.

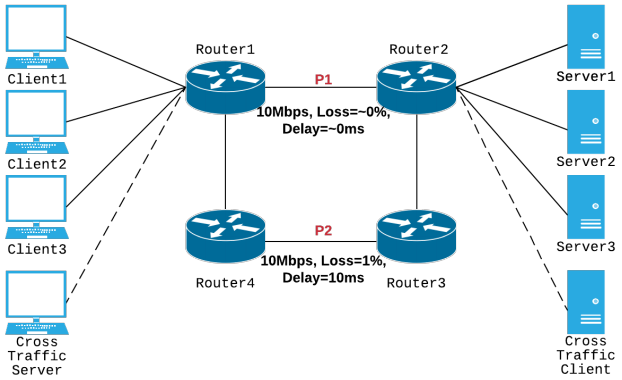


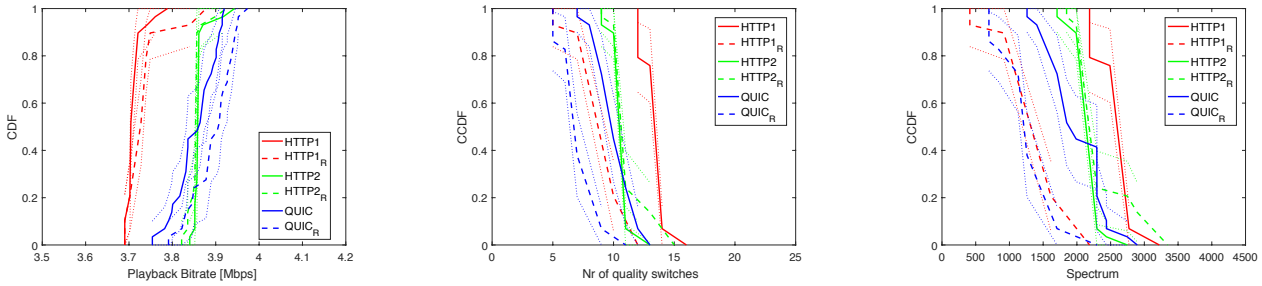
Figure 6: Cloudlab topology used for controlled experiments

consists of several different racks of varying compute and storage resources designed to provide isolated performance. The topology shown in Fig. 6 consists of four clients and four servers connected by two paths **P1** and **P2** with the default set to **P1** unless stated otherwise. All nodes run vanilla Ubuntu 14.04 where all TCP related experiments use TCP Cubic. In order to account for statistical variance, every experiment in the controlled environment is repeated 30 times. For the single client experiments, we use *Client1* and *Server1* as the default pair and include other server and client pairs for parallel client cases.

4.1.1 Single Client: Rate Limiting with UDP. In order to systematically compare the performance of HTTP/1.1, HTTP/2 and QUIC in a controlled environment, we use the `Iperf`[7] application to generate competing UDP traffic (denoted cross traffic) of varying amplitudes. The first set of experiments consists of repeating a stepwise variation of cross traffic where the duration of each step is 11s and varies as follows: {0-11s: 0Mbps, 12-23s: 3Mbps, 24-35s: 6Mbps, 36-55s: 9Mbps, 56-67s: 6Mbps, 68-79s: 3Mbps, 80-91s: 0Mbps} (then the pattern repeats until t=300s). Fig. 7 shows the CDF and CCDF along with 95% confidence intervals for upper and lower bounds of the QoE metrics described at the beginning of this section. In Fig. 7a, we observe that QUIC clients have the highest average quality bitrate or *AQB* when compared to both HTTP/1.1 and HTTP/2. It is also worth noting that other QoE metrics such as number of quality changes (#*QS*) and the Spectrum, *H*, are significantly improved with the use of QUIC retransmissions. Figure 8 shows results for the "W" cross traffic case where we use the `Iperf` application to generate competing UDP cross traffic that creates a "W" shaped bottleneck bandwidth and varies as follows: {0-20s: 9Mbps, 21-40s: 5Mbps, 41-60s: 9Mbps, 61-80s: 0Mbps} (then the pattern repeats until t=300s). Although, in terms of *AQB*, #*QS* and *H*, HTTP/2 clients appear to experience the best QoE, we observed that the clients also experience a relatively high rebuffering ratio, *RB*, of 4% while using HTTP/2 for ABR streaming. Since it is well known that the foremost objective of any ABR client streaming algorithm is to eliminate or reduce rebuffering, we conclude that QUIC, especially with the use of segment retransmissions, also performs significantly better than HTTP/1.1 and HTTP/2 for the "W" cross traffic case.

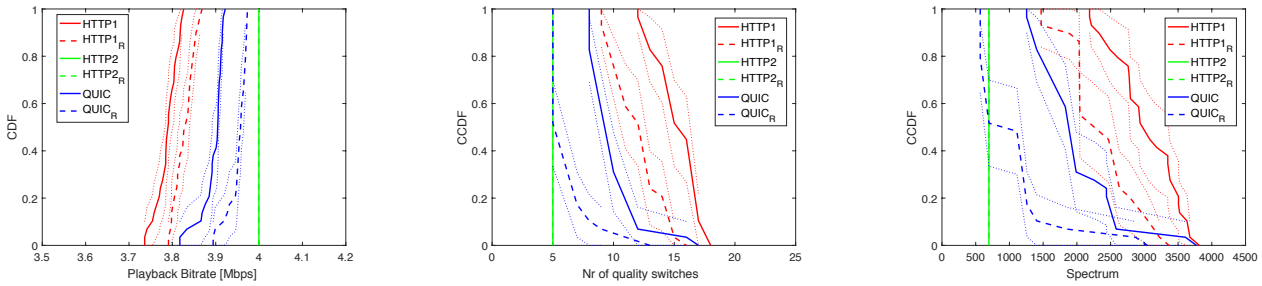
4.1.2 Single Client: Re-ordering and HOL. Since packet reordering in the Internet is not uncommon [17], protocols for ABR streaming should be robust in the face of such reordering. Here, we study the ability of HTTP1.1, HTTP/2, and QUIC to recover from reordering of packets. This is the only experiment where we use the second path (denoted **P2** in Fig. 6) to carry video streams. In order to induce re-ordering of packets, we switch between a low latency, low loss path, **P1**, and a high latency, high loss path, **P2**, every second using SDN, namely the OpenFlow [22] implementation, which provides fine-grained, dynamic traffic engineering for application packets. As shown in Fig. 6, **P2** is characterized by 1% loss and

[7] https://iperf.fr/iperf-doc.php



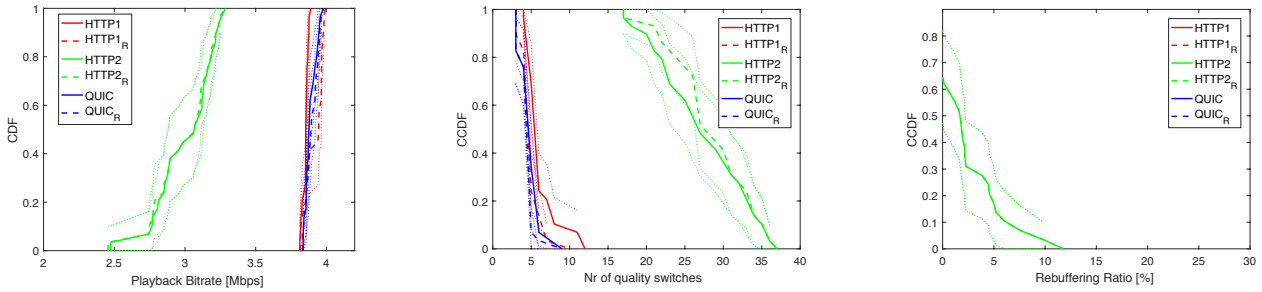
(a) (b) (c)

Figure 7: Single Client Measurements - Rate Limited with UDP-Staircase cross traffic. QUIC has a significantly better overall Quality of Experience compared to HTTP/1.1 and HTTP/2, which is further improved by retransmissions. Note, subscript "R" denotes ABR segment retransmissions.



(a) (b) (c)

Figure 8: Single Client Measurements - Rate Limited with UDP-W cross traffic. QUIC has a significantly better overall QoE compared to HTTP/1.1. Although HTTP/2 sessions appear to be having a higher QoE, all clients experience 4% rebuffering. Note, subscript "R" denotes ABR segment retransmissions.



(a) (b) (c)

Figure 9: Single Client Measurements - Re-ordering and Head-of-Line Blocking. Re-ordering has an adverse effect on HTTP/2 causing significant degradation of QoE metrics, especially with respect to rebuffering which can be as high as 10% in spite of selecting lower quality bitrates as seen from (a). Note, subscript "R" denotes ABR segment retransmissions.

10ms delay implemented using `tc`[8] and `netem`[9] utilities. For the experiments presented in Sect. 4.1.1, we find that HTTP/2 is either comparable or marginally worse than HTTP/1.1 and QUIC. In the case of packet reordering (shown in Figure 9), we see that HTTP/2 performs significantly worse than QUIC and HTTP/1.1 . Not only is the *AQB* significantly lower with a high variation between runs, but also the rebuffering is as high as 10% where over 60% of clients experience an *RB* of 2.5%. Further analysis using the `tshark`[10] utility reveals that a HTTP/2 session experiences 9.5% fast TCP retransmits. In comparison, HTTP/1.1 experiences 7.1%, and QUIC sessions experience no UDP retransmissions since they use NACKs (c.f. Sect. 2.1). More details on the `tshark` data can be found in [11] Additionally, QUIC uses a higher initial congestion window size=32

[8]http://lartc.org/manpages/tc.txt
[9]http://man7.org/linux/man-pages/man8/tc-netem.8.html
[10]https://www.wireshark.org/docs/man-pages/tshark.html

(the Linux default for TCP is 10) and also grows the window more aggressively, thus, allowing more unacknowledged bytes in flight. This results in a more reliable download rate measurement and a stable buffer level for the ABR client and consequently, a reduction in the quality variations #*QS* as observed in Fig. 9b.

4.1.3 Parallel Clients: Competing Traffic. For this experiment we use the additional client and server pairs (denoted as *Client2*/*Server2* and *Client3*/*Server3* in Figure 6) to initiate three simultaneous sessions of QUIC-based SQUAD clients. Although all three clients enjoy a smooth playback experience without rebuffering as shown in Figure 10, we note that the bandwidth sharing can result in unfair behavior in the case of ABR streaming sessions. This is contrary to the analysis presented by the authors of [19] where they observe that QUIC flows are fair to each other but only unfair to TCP flows when downloading a file. While we similarly observe that QUIC does tend to "starve out" TCP flows, we note that ABR streaming over QUIC with the use of retransmissions can result in unfair behavior for competing ABR streams since multiplexing due to retransmissions can occur at different points throughout the streaming session. In order to corroborate this analysis, we present the percentage of retransmissions in Table 1, which shows that the three clients experience varying number of ABR segment retransmissions per run. Since these retransmissions occur asynchronously, the clients observe different buffer levels and rate measurements throughout a streaming session. We also perform similar experiments with three HTTP/2 clients and observe that HTTP/2 shows a nearly equal distribution of *AQB* and closer inspection reveals that the *AQB* of *Client1* is 0.5Mbps higher on average as compared to the other two clients. Since TCP is more conservative about setting the initial congestion window size and has a less aggressive window growth it enables all three clients to have a "fair" share of the bottleneck bandwidth. Further details on the TCP-based experiments can be found in [11].

	Client1	*Client2*	*Client3*
Average %Retransmissions	0.8±1.3	1.7±1.3	1.0±0.9

Table 1: ABR Segment Retransmissions for three parallel QUIC clients

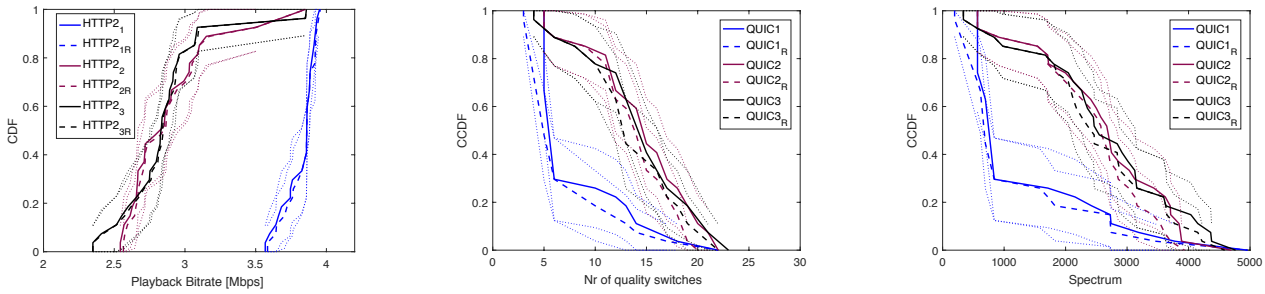
4.2 Internet

For the Internet measurements, we use Amazon EC2 virtual machines in Mumbai, India and Oregon, USA as servers and a client in the UMass Amherst campus network to perform inter-continental and intra-continental measurements, respectively. Here, we repeat each experiment 60 times to account for increased network variations in an uncontrolled environment. Since the bottleneck bandwidth during off-peak hours can be high, we use a different video dataset with higher qualities, `RedBull` [21], and modify the MPD to contain the following bitrates {0.10, 0.15, 0.20, 0.25, 0.30, 0.40, 0.50, 0.70, 0.90, 1.20, 1.50, 2.00, 2.50, 3.00, 4.00, 5.00, 6.00}Mbps for a video duration of 300s and a segment duration of 2s. Figure 11 presents results for measurements "in the wild" over inter-continental links from an EC2 web server located in India. The average quality bitrate (in Fig. 11a) is significantly higher for QUIC than HTTP/2 and HTTP/1.1. Fig. 11b shows that #*QS* is also reduced with the use of QUIC and HTTP/1.1 retransmissions indicating an overall high QoE. Table 2 shows QoE metrics for similar measurements conducted with the server located at EC2 in Oregon. Here, it is worth mentioning that all QoE metrics are comparable for HTTP/1.1 and QUIC where QUIC is marginally better than HTTP/1.1, but are significantly improved over HTTP/2 (for example, the average bitrate *AQB* is less than half of that obtained with HTTP/1.1 and QUIC). Since Internet traffic is predominantly comprised of TCP flows, these results further reinforce the observations made in Sect. 4.1.2 for high delay, high loss paths with competing TCP traffic. Our results show that the use of QUIC results in better QoE in the case of inter-continental as well as intra-continental links, while the advantage compared to HTTP/1.1 is more significant in case of the former.

5 RELATED WORK

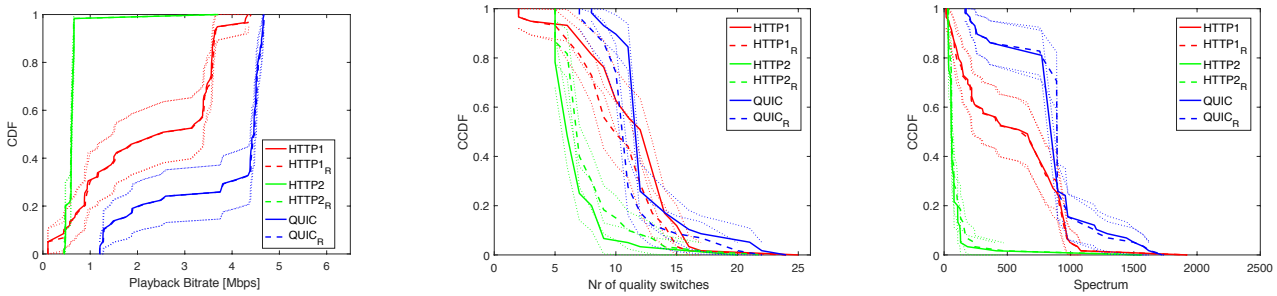
A recent paper by Google [20] provides a detailed analysis of an Internet-scale deployment of QUIC. They specifically look at latency and rebuffer rate in order to understand the performance implications of QUIC for video streaming over YouTube. Timmerer et al. [29] evaluate ABR streaming over QUIC for varying network latencies and show that there is no significant benefit to QoE streaming with the use of QUIC. In [28], a demonstration by Szabó et al. provides a new congestion control mechanism for QUIC that aggressively varies download rate according to a buffer-based priority level assigned by the ABR streaming client. Carlucci et al. [9] present results that compare TCP and QUIC under varying network conditions and buffer size. In [19], Kakhki et al. perform a detailed analysis of QUIC under varying network conditions to investigate the benefits of using QUIC for applications such as web browsing and video streaming over YouTube. The authors of [7] also compare the performance of several rate adaptive DASH players including QUIC and conclude that QUIC is more aggressive compared to TCP. The authors of [14] devise and deploy an SDN approach to to improve the QoE of ABR streaming by monitoring MPTCP retransmissions where their system dynamically switches between network paths and protocols to mitigate re-ordering effects. While we similarly compare the performance of TCP (using HTTP/1.1 and HTTP/2) with QUIC, our work is more focussed on the potential benefits that QUIC can provide for video streaming especially with respect to retransmitting video segments in higher qualities. Similar experiments are performed by the authors of [16], where they use the multiplexing feature of HTTP/2 to simultaneously request multiple qualities of a segment. While retransmissions can be regarded as an additional burden on the available bandwidth we note that recent works such as [31] suggest different types of redundant transmission to provide higher QoS. In contrast to [16] and [31], we only invoke retransmissions in a systematic way, thereby guaranteeing an improvement in QoE while also minimizing the consumption of additional bandwidth. Moreover, in order to analyze the implications of specific network conditions that affect ABR video streaming, we design, develop and prototype such a system in a nearly isolated, controlled testbed environment.

Legacy protocols that perform adaptive bitrate video streaming over UDP include systems such as Real-time Transport Protocol (RTP) [12] and Stream Control Transport Protocol (SCTP) [27]. Similar to QUIC, SCTP also allows multiplexing of multiple chunks



(a) (b) (c)

Figure 10: Parallel Client Measurements - Three QUIC Clients. Competing QUIC clients show an unfair behavior where two clients experience relatively similar QoE but one client has a significantly better QoE than others.



(a) (b) (c)

Figure 11: Internet Measurements - ABR streaming is performed over inter-continental links with the server at Amazon EC2 in India and the client on the US East Coast. QUIC far outperforms HTTP/1.1 and HTTP/2 in terms of QoE, i.e., provides significant improvement in Average Quality Bitrate while providing comparable reduction in the number of quality switches.

	AQB (Mbps)	AQB_R (Mbps)	$\#QS$	$\#QS_R$	H	H_R	RB_R(%)
Internet: HTTP/1.1	5.31±0.1	5.66±0.1	8.48±1.4	3.82±2.1	490±213	242±312	0
Internet: HTTP/2	2.12±0.6	2.13±0.6	9.09±2.6	6.98±2.5	552±280	447±255	0±10.8
Internet: QUIC	5.31±1.9	5.44±0.2	7.91±1.8	5.81±1.7	445±299	351±273	0

Table 2: ABR Quality of Experience over the Internet: Amazon EC2 Oregon - US East Coast

into one packet and avoids HOL blocking, thus, allowing unordered delivery to the application layer. Unlike QUIC, SCTP implements congestion control according to the TCP *NewReno* specification which uses Selective Acknowledgement (SACK) for loss recovery. Another example of an ABR protocol over UDP is the Video Transport Protocol (VTP) which was designed and evaluated by Balk et al. [8]. In this work, the authors employ a form of congestion avoidance where the sending rate at the server is increased by a single packet for every RTT measurement. This design is different from the AIMD congestion control employed by TCP and QUIC since it eliminates the effect of slow start and attempts to provide an accurate estimate of the available bandwidth in the network. Some drawbacks of this approach are the requirement of two UDP sockets for every connection and the use of Berkeley Packet Filters to collect timestamps at the server and client for every video stream, thus, reducing both performance and scalability of the system. Although there are a number of server push approaches such as [16] and [34] that have been proposed for HTTP/2, adapting such systems for retransmissions would not scale since the computation and storage overhead incurred on the server per individual client connection would render such an approach infeasible.

6 CONCLUSION

In this work, we conduct systematic experiments to analyze the performance implications of various HTTP/2 transport layer candidates on ABR streaming systems, particularly with respect to ABR segment retransmissions. We leverage the multiplexing feature of QUIC and HTTP/2 in order to efficiently implement parallel retransmissions in a higher quality with the objective of maximizing average quality bitrate while also minimizing bitrate variations throughout the duration of a streaming session. We use a nearly isolated testbed setup in CloudLab and measurements "in the wild" to show that QUIC retransmissions provide a significantly better QoE than TCP in high latency, high loss networks while exhibiting comparable QoE in low latency, low loss networks.

REFERENCES

[1] NetFlix. http://www.netflix.com. Accessed: 2018-03-10.

[2] Packet Pacing in QUIC. https://groups.google.com/a/chromium.org/forum/#!topic/proto-quic/fRk0I98VSMk. Accessed: 2018-03-10.

[3] QUIC, a multiplexed stream transport over UDP. https://www.chromium.org/quic. Accessed: 2018-03-10.

[4] Usage of http/2 for websites. https://w3techs.com/technologies/details/ce-http2/all/all. Accessed: 2018-03-10.

[5] YouTube. https://www.youtube.com. Accessed: 2018-03-10.

[6] Caddy web server. https://caddyserver.com/, Accessed 3-9-2017.

[7] 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.

[8] A. Balk, M. Gerla, M. Sanadidi, and D. Maggiorini. Adaptive mpeg-4 video streaming with bandwidth estimation: Journal version. *vol*, 44:415–439, 2003.

[9] 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*, SAC '15, pages 609–614, New York, NY, USA, 2015. ACM.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1, 1999.

[11] A. for double-blind review. TR - Improving QoE of ABR Streaming Sessions through QUIC Retransmissions. https://drive.google.com/open?id=1E8PfspsHTTNMfRgA5iuY_SsiiOi53iio, 2018.

[12] R. F. H. Schulzrinne, S. Casner and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications, 2017.

[13] S. Ha, I. Rhee, and L. Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[14] B. Hayes, Y. Chang, and G. Riley. Omnidirectional adaptive bitrate media delivery using mptcp/quic over an sdn architecture. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–6, Dec 2017.

[15] T. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, timid, and unstable: Picking a video streaming rate is hard. In *Proc. of IMC*, pages 225–238, 2012.

[16] R. Huysegems, J. van der Hooft, T. Bostoen, P. Rondao Alface, S. Petrangeli, T. Wauters, and F. De Turck. Http/2-based methods to improve the live experience of adaptive streaming. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 541–550. ACM, 2015.

[17] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Measurement and classification of out-of-sequence packets in a tier-1 ip backbone. *IEEE/ACM Transactions on Networking*, 15(1):54–66, Feb 2007.

[18] P. Juluri, V. Tamarapalli, and D. Medhi. SARA: Segment-aware Rate Adaptation Algorithm for Dynamic Adaptive Streaming over HTTP. In *IEEE ICC QoE-FI Workshop*, 2015.

[19] 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*, IMC '17, pages 290–303, New York, NY, USA, 2017. ACM.

[20] 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*, SIGCOMM '17, pages 183–196, New York, NY, USA, 2017. ACM.

[21] S. Lederer, C. Müller, and C. Timmerer. Dynamic adaptive streaming over HTTP dataset. In *ACM MMSys*, pages 89–94, 2012.

[22] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[23] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai network: A platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, Aug. 2010.

[24] R. Ricci, E. Eide, and The CloudLab Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *USENIX ;login:*, 39(6), Dec. 2014.

[25] I. Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia*, 18(4):62–67, April 2011.

[26] D. Stenberg. Http2 explained. *SIGCOMM Comput. Commun. Rev.*, 44(3):120–128, July 2014.

[27] R. Stewart, Q. Xie, K. Morneault, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol, 2017.

[28] G. Szabó, S. Rácz, D. Bezzera, I. Nogueira, and D. Sadok. Media qoe enhancement with quic. In *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on*, pages 219–220. IEEE, 2016.

[29] C. Timmerer and A. Bertoni. Advanced transport options for the dynamic adaptive streaming over http. *arXiv preprint arXiv:1606.00264*, 2016.

[30] S. I. ULC. Global Internet phenomena report 2016, 2016.

[31] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Low latency via redundancy. In *Proc. of CoNEXT*, pages 283–294, 2013.

[32] C. Wang, D. Bhat, A. Rizk, and M. Zink. Design and analysis of qoe-aware quality adaptation for dash: A spectrum-based approach. *ACM Trans. Multimedia Comput. Commun. Appl.*, 13(3s):45:1–45:24, July 2017.

[33] C. Wang, A. Rizk, and M. Zink. SQUAD: A Spectrum-based Quality Adaptation for Dynamic Adaptive Streaming over HTTP. In *Proc. of MMSys*, pages 1:1–1:12. ACM, 2016.

[34] M. Xiao, V. Swaminathan, S. Wei, and S. Chen. Evaluating and improving push based video streaming with http/2. In *Proceedings of the 26th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, page 3. ACM, 2016.

[35] M. Zink, J. Schmitt, and R. Steinmetz. Layer-encoded video in scalable adaptive streaming. *IEEE Transactions on Multimedia*, 7(1):75–84, Feb 2005.