Not so QUIC: A Performance Study of DASH over QUIC

Divyashri Bhat[1], Amr Rizk[2], Michael Zink[1]

[1]University of Massachusetts Amherst, [2]Technische Universität Darmstadt
dbhat@ecs.umass.edu,amr.rizk@kom.tu-darmstadt.de,mzink@ecs.umass.edu

ABSTRACT

Despite known QoE shortcomings, Dynamic Adaptive Streaming over HTTP (DASH) has been tied with TCP for many years now. The advent of HTTP/2 powered by transport protocols such as QUIC provides an excellent opportunity to revisit adaptive bitrate streaming with respect to QoE. QUIC promises improved congestion control, zero-RTT connection establishment and multiplexing logical streams. In this work, we adapt state-of-the-art DASH players with buffer-based and hybrid (rate/buffer-based) quality adaptation logic to use QUIC. Our main focus lies in contrasting the QoE performance of DASH algorithms running on top of QUIC versus TCP in various environments. Interestingly, we find through testbed and Internet measurements that QUIC does not provide a boost to current DASH algorithms but instead a degradation in the chosen quality bitrates.

KEYWORDS

Video streaming protocols, DASH, QUIC, ABR streaming, Video quality metrics, QoE

1 INTRODUCTION

Adaptive bitrate (ABR) streaming has become the defacto streaming standard for video on demand platforms such as Netflix [1] and Youtube [2]. With more than 70% of the peak hour US Internet traffic [23], video streaming has become *the* killer-application of the Internet today. State-of-the-art ABR video streaming solutions, i.e., Dynamic Adaptive Streaming over HTTP (DASH) are, however, stuck in an HTTP/TCP setting that has been shown to possess substantial drawbacks with respect to the user's Quality-of-Experience (QoE) [10, 24]. These drawbacks stem mainly from a dual control loop on the application and the transport layer, which produces, e.g., an ON-OFF transmission behavior of video segments leading to short consecutive TCP transmissions with known fairness issues. Despite these shortcomings, the state-of-the-art setting with ABR streaming over HTTP and TCP had a very positive impact on the growth of video streaming applications over the Internet. The use of HTTP/TCP has helped video streaming, e.g., *(i)* to overcome diverse video player specifications by utilizing web browsers, *(ii)* to adapt to available bandwidth changes, and *(iii)* to transcend firewalls and ISP middleboxes.

The advent of HTTP/2 marks a new wave of radical improvements to HTTP that enable, e.g., the multiplexing of logical streams, server push, and request prioritization. These properties are provided, e.g., by Google's QUIC protocol [16], that is a transport layer candidate for HTTP/2. QUIC promises improved congestion control over UDP, fast connection establishment, and seamless connection migration. This paper aims to answer the following questions:

(1) What is the impact of QUIC on QoE?
(2) How should state-of-the-art adaptive bitrate streaming be built to leverage the benefits provided by QUIC?

To answer these questions, we study the QoE performance of DASH streaming algorithms within a testbed environment both for traditional HTTP over TCP settings and using QUIC. We consider recently presented DASH quality adaptation algorithms that are mainly categorized as throughput- and/or playout buffer-based techniques. These algorithms have been designed and fine tuned for DASH over TCP to provide high QoE. It remains open how these algorithms react and maybe need to be modified when used on top of QUIC. Here, we evaluate the QoE performance of the different settings using quantitative metrics from [25]. We also evaluate QoE fairness aspects when multiple streaming applications compete for network resources. In addition, we perform a set of experiments in the public Internet to evaluate the performance of DASH over QUIC in an uncontrolled environment with background traffic.

This paper is structured as follows: In Section 2, we briefly introduce QUIC and discuss the potential integration of QUIC into minimally modified DASH clients. Section 3 describes our experiments that are designed to compare the performance of QUIC-enabled and TCP-enabled DASH clients under varying network conditions. In Section 4 and Section 5, we include an analysis of our observed measurements and a discussion of potential benefits of incorporating QUIC transport in DASH clients, respectively. Section 6 presents an overview of other relevant works that study alternative transport protocols for DASH clients.

2 QUIC VS. TCP

In this section, we shed light on the main differences between TCP and QUIC protocols and how we expect these differences to impact the performance of DASH clients. Here, we summarize the features of QUIC obtained from [20] that can be included in DASH players with little or no modification and test them in our current evaluation. In Sect. 5 we provide a detailed discussion of the features of QUIC that could potentially improve the performance of DASH video

NOSSDAV'17, Taipei, Taiwan
 978-1-4503-5003-7/17/06...$15.00
DOI: http://dx.doi.org/10.1145/3083165.3083175

delivery systems with significant modification to ABR client and server systems.

2.1 Handshake

TCP requires a 3-way handshake and thus, requires 1.5RTT before any data request is received at the server. QUIC, on the other hand, requires roughly 0.5RTT before a data request is received at the server. For ABR clients, the faster handshake implemented in QUIC should be beneficial in reducing the startup latency.

2.2 Congestion Control

The default congestion control algorithm implemented in QUIC is similar to that of TCP Cubic [9] 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 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 server. The use of NACKs allows much faster loss recovery and could mean high reduction in the rebuffering ratio for DASH clients.

In this work, we evaluate the potential benefits of using the congestion control provided by QUIC for various ABR client algorithms.

3 EXPERIMENT SETUP

In this section, we describe the measurement setup which is used to evaluate the performance of DASH over QUIC. First, we provide a short overview on DASH and a set of state-of-the-art quality adaptation algorithms followed by a description of the setup for the controlled measurements in the CloudLab testbed and the setup used for our Internet-based measurements.

3.1 DASH Adaptation Algorithms

DASH is an MPEG adaptive streaming standard developed for the streaming of media content from web servers via HTTP [19]. In case of DASH, the video source content is chunked into small (between 2 and 10 secs. in duration) segments and each is encoded at multiple bit rates.

The DASH standard does not exactly specify how quality adaptation should be performed which led to a series of existing work on this topic. In the following, we describe three state-of-the-art algorithms, which we have chosen for the evaluation described in Sect. 4.

BBA-2 [11]: BBA-2 is implemented as part of the Python DASH client emulator accompanying [12]. In a nutshell, the algorithm defines a class of functions that map current buffer occupancy to a quality bitrate (denoted rate map) to avoid unnecessary rebuffering and maximize the average video rate. This algorithm was part of a wide-scale Netflix experiment presented in [11].

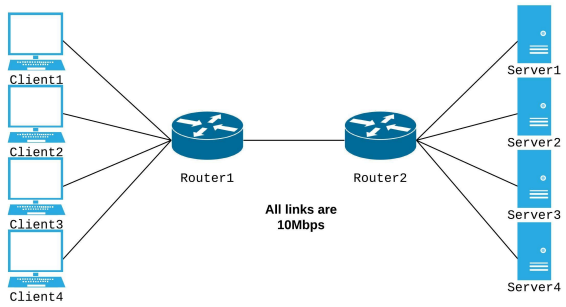

Figure 1: Cloudlab topology used for controlled experiments

SQUAD [24]: This hybrid adaptation algorithm is based on the spectrum metric for QoE [25]. It uses a combination of buffer and rate-based quality adaptation that accounts for the dynamics of TCP on different time scales. SQUAD avoids sudden quality changes under volatile bandwidth conditions by sacrificing buffer filling if the requested quality bitrate is sustainable.

BOLA [17]: BOLA is a buffer-based quality adaptation algorithm. It uses a Lyapunov technique for renewal processes to decide on the quality of the next segment to be fetched. While BOLA(U) aims to maximize a playback utility metric which is a weighted combination of quality bitrate and smoothness (average rebuffering time), BOLA(O) tries to minimize the oscillation in the average quality bitrate by sacrificing buffer filling without the risk of rebuffering to maintain the previously downloaded quality.

We have chosen these three algorithms since they have been recently published and present different approaches (buffer, rate, and combined) for quality adaptation.

3.2 Testbed Setup

For our controlled experiments, we use Cloudlab [15] which is a geographically distributed testbed for the development, deployment, and validation of cloud-based services. The CloudLab infrastructure consists of several different racks of varying compute and storage resources designed to provide isolated performance. The butterfly topology shown in Fig. 1 consists of four clients and four servers connected through a bottleneck link of 10Mbps. All nodes are bare metal machines that run vanilla Ubuntu 14.04 where all TCP related experiments use TCP Cubic with the default Explicit Congestion Notification (ECN) setting, i.e, we enable ECN when requested by incoming connections but do not request ECN on outgoing connections. The server nodes run a Caddy [3] server with the experimental QUIC mode enabled such that the clients can stream DASH videos either over TCP or QUIC. The client nodes run the ABR algorithms described above which are implemented in a Python-based DASH player [12].

For the following experiments, the four clients shown in Fig. 1 simultaneously stream the first 300s of the *BigBuckBunny* video [13] where each client is started with a time offset of 1s after the previous one.

3.2.1 Experiment 1: Parallel server. For the first set of experiments, each client streams from a unique server, i.e.,

Client1 streams a video from *Server1*, *Client2* streams the same video from *Server2* and so on. This experiment is designed to analyze the performance of the congestion control mechanism implemented by QUIC and test its ability to provide fairness to all streaming clients. We run a series of 40 measurements (10 for each algorithm) where two clients, i.e, *Client1* and *Client2* use TCP at the transport layer while the other two clients, i.e., *Client3* and *Client4* stream the same video using QUIC.

3.2.2 Experiment 2: Single server. For the second set of experiments, each client streams from the same server, i.e., *Client1 – Client4* all stream the same video from *Server1*. We perform a series of 80 measurements for this experiment. We have eight different setups (BOLA(O), BOLA(U), SQUAD, BBA each over TCP and QUIC) and for each 10 runs are performed. Through this experiment, we observe the loss recovery mechanism (discussed in Sect. 2.2) implemented by QUIC and its effect on various ABR client algorithms. We emulate various delay and loss combinations in the bottleneck link between *Router1* and *Router2* shown in Fig. 1 using the Network Emulator (*NetEm*) tool.

3.3 Internet Setup

In order to evaluate the performance of QUIC as a transport protocol for DASH in the real-world, we run experiments over the public Internet. In this scenario, the server is running on an Amazon EC2 instance in a data center in Northern California, while we use three different locations for the clients. In the first case, the client is connected via WiFi on a campus network. The second client is connected via WiFi to residential access network (Comcast). Both clients are located in the Northeastern US. The third client is connected via a wired campus Network located in Southern California. The clients run the algorithms that were also used in the controlled experiment case. We use the RedBull dataset [13] with 17 bitrates ranging from $100Kbps - 5.9Mbps$ for our measurements. Each measurement consists of a 5 min video stream repeated 30 times for statistical evaluation.

4 EVALUATION

In this section we present the results obtained for QUIC and TCP in various client and network load scenarios. The metrics we use for evaluation are as follows:

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.

Spectrum (H) [25]: 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:

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

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

4.1 Testbed

Here, we analyze the performance of QUIC vs. TCP in a controlled testbed environment and depict the average QoE metrics with confidence intervals for all measurements. For the first set of experiments consider Fig. 2 which represents the QoE performance in the case of 2 TCP flows and 2 QUIC flows that share the same bottleneck link of 10Mbps as depicted in Fig.1. It is interesting to note that DASH players utilizing TCP possess an average quality bitrate (denoted as *Chosen Rate*) that is significantly higher than in the QUIC case as seen in Fig. 2a. Other QoE metrics that look at the number of quality changes $\#QS$ (Fig. 2b) and the spectrum H (Fig. 2c) indicate similar quality degradation in the case of the two QUIC clients as compared to the TCP case. These results tell us that DASH clients that use TCP are more aggressive in downloading higher qualities as compared to the ones that use QUIC.

Figure 3 presents similarly interesting insights on the behavior of QUIC in the case where all 4 clients stream from a single server. These experiments evaluate the performance of QUIC for constant delay and loss conditions in the link between *Router1* and *Router2*. For the average quality bitrate AQB shown in Fig. 3a, QUIC possesses a slightly lower value, although it is not as significant as in the case of the parallel experiments. What we observe here is that AQB is almost similar in the case of all algorithms that use some form of buffer-based segment download but is significantly worse for QUIC in the case of SQUAD, which is a rate-based algorithm that is specifically optimized for variations in TCP download rates. In the case of the number of quality changes $\#QS$ and the spectrum H, BBA-2 seems to benefit from using QUIC to download segments. The main difference between BBA-2 and the other algorithms is the fact that in the slow-start or initial phase of the algorithm, the client tends to download a quality that is higher than the measured rate in order to provide the user with maximum average quality which indicates that only significantly larger RTTs will cause enough buffer drain to motivate the BBA-2 client to switch to a lower quality.

Since the QoE metrics for QUIC shown in Fig. 3 are noticeably better than the results in Fig. 2, we deduce that QUIC ABR streams compete better with each other than with a mix of TCP and QUIC streams.

4.2 Internet

In this set of experiments we compare the QoE performance of the DASH algorithms introduced in Sect. 3.2 when streaming to different client locations as described in Sect. 3.3. Figure 4 shows the corresponding results. In Fig. 4a we depict the average quality bitrate chosen by the different adaptation algorithms. Note that, in general, clients that are placed near to the streaming server (UCLA) with a wired Internet

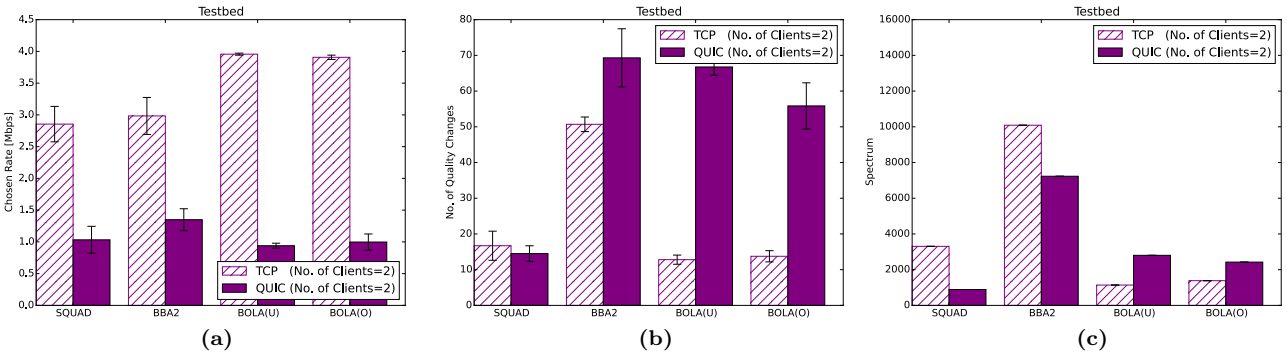

Figure 2: Testbed measurements: Hybrid and buffer-based DASH quality adaptation algorithms over QUIC vs TCP. All quality adaptation algorithms perform significantly better in the case of the two TCP clients as compared to the two QUIC clients. The TCP DASH clients are more resilient to dissimilar competing traffic than the QUIC DASH clients.

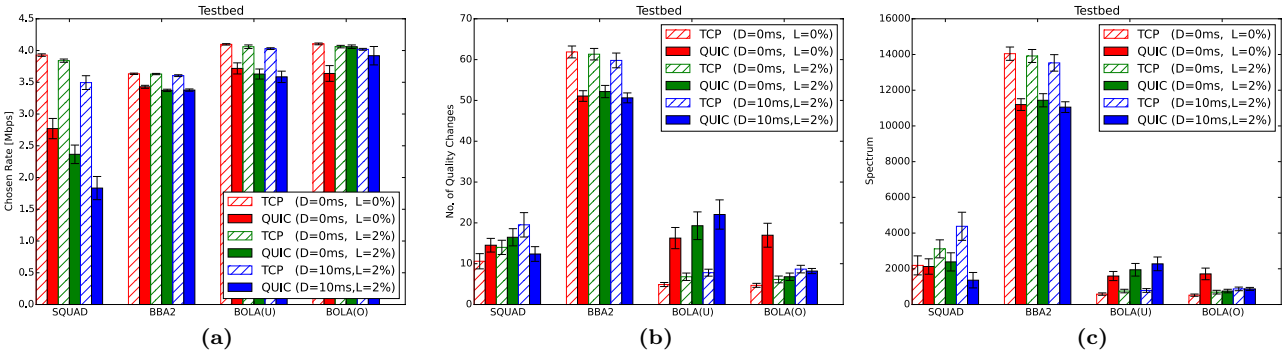

Figure 3: Testbed measurements: Hybrid and buffer-based DASH quality adaptation algorithms over QUIC vs TCP. Fixed loss and delay emulations significantly degrade the average quality for the rate-based algorithm over QUIC whereas buffer-based algorithms are more resilient to average quality degradations over both transport protocols.

connection possess the highest streaming QoE performance with unnoticeable differences between the QUIC and TCP cases. More interesting are the cases when the client is located at the US east coast while streaming over WiFi. Here, we find that QUIC mostly provides a significantly worse quality bitrate across adaptation algorithms and scenarios. This underlines the testbed results from Sect. 3.2. We attribute this phenomenon to the fact that in the Internet, where QUIC is likely to be competing with more TCP streams than other QUIC streams, it takes longer than TCP to fetch the same segments. On average, this leads to a lower playout buffer filling in case of QUIC which is detrimental for the requested quality bitrates. Hence, we observe that especially buffer-based quality adaptation algorithms are highly affected by this phenomenon. A closer look at the spectrum in Fig. 4b and the CDFs of the magnitude of quality changes (not depicted here) shows, however, that DASH over QUIC has in most cases a lower magnitude of quality switches than DASH over TCP. Interestingly, we only observe in Fig. 4d rebuffering events, i.e., video stalling, when DASH is running on top of QUIC. This lets us conjecture that buffer-based quality adaptation algorithms running on top of QUIC use *dangerously* low buffer fillings that risk multiple video stalling events.

5 DISCUSSION

In general, we observe that for existing DASH algorithms under varying network conditions, TCP provides considerably better performance than QUIC. We attribute this to the fact that most quality adaptation algorithms are optimized to work well with TCP and therefore, they do not fully utilize the features provided by QUIC. In this section, we discuss some of these features and the potential benefit they could bring to suitably modified DASH clients.

5.1 Disabled Head-of-Line (HOL) blocking

The negative impact of HOL blocking on application performance, especially in the case of live video, has been evaluated by previous works such as [14]. HOL blocking arises mainly in TCP due to the in-order delivery guarantee but UDP does not provide any similar guarantee. Unlike TCP, which is restricted to a single stream per connection, QUIC utilizes multiple streams per connection to quickly obtain data from the server. This means that the loss of data on one stream marginally affects other streams. This potentially provides the following performance gains for DASH:

- Using QUIC's inherent ability to simultaneously download over multiple streams in a single connection, multiple qualities of a segment are obtained in

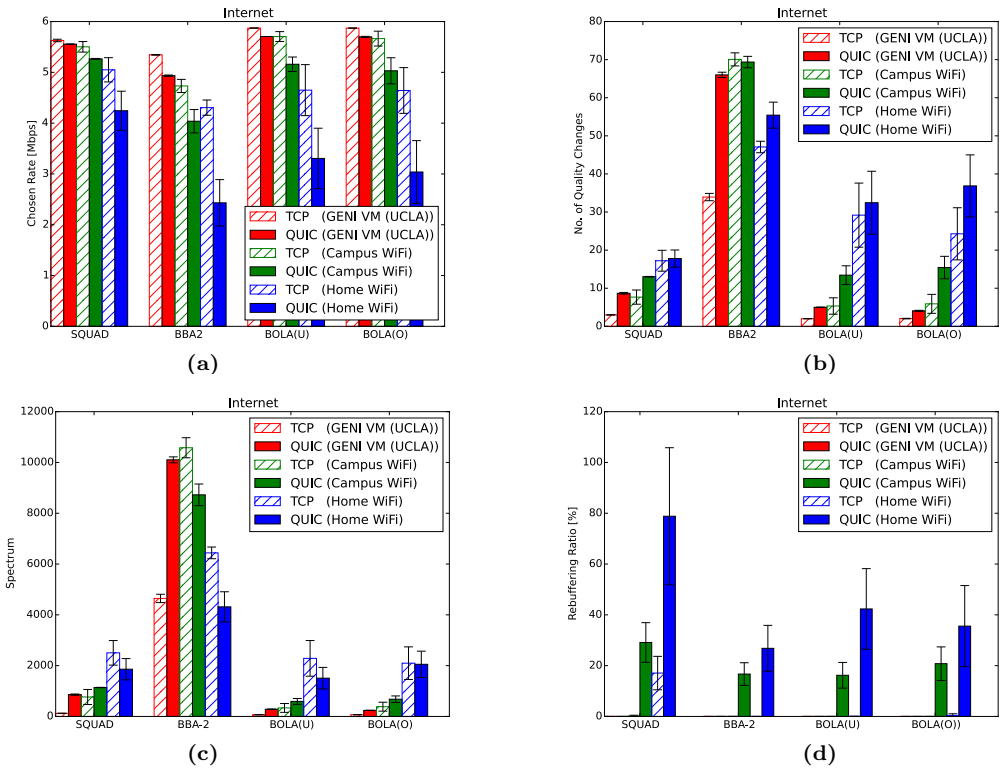

Figure 4: Internet measurements: Hybrid and buffer-based DASH quality adaptation algorithms over QUIC vs TCP. QUIC consistently provides lower quality bitrates, however, with lower quality magnitude variations. Buffer-based quality adaptation algorithms are affected the most by low playout buffer fillings through QUIC.

parallel. Due to the sequential nature in which segments are requested over a TCP connection, emulations of simultaneous downloads of multiple qualities provide little performance gain. DASH algorithms such as BOLA [17] sequentially abandon unsustainable downloads and request lower qualities in order to avoid excessive buffer drain. The multiplexing feature of QUIC would greatly benefit algorithms that perform abandonment and retransmissions.

- Multiple segments are downloaded on separate streams using the same QUIC connection. Since network conditions tend to vary over time, especially in the case of mobile clients, existing DASH algorithms can be modified to download a train of segments in parallel QUIC streams when a stable network state is detected in order to maximize average quality bitrate. Lessons for implementing this functionality can be learned from existing Multi-Path TCP (MPTCP) approaches that allow segments to be downloaded simultaneously using multiple interfaces [7].

5.2 Pacing

The pacing feature of QUIC could have considerable benefits for the DASH server in terms of load balancing, fairness and QoE gains. Currently, DASH servers that use TCP are limited by the congestion control feature of standard TCP implementations and are automatically rate-controlled. However, with the use of QUIC, the server pacing will allow a more continuous streaming of the segment within connections that would offer the following advantages:

- If a client requests multiple qualities for a single segment, the server could pace the streams to deliver segments at regular intervals instead of using AIMD in order to avoid rebuffering at the client.
- If a client requests multiple segments over the same connection via multiple streams the server could implement a decreasing pacing rate for segments requested depending on how soon they are required for playback.

5.3 Eliminate Congestion Control Redundancy

One of the main restrictions of using TCP is that the transport mechanism cannot be controlled by requirements of various applications. While it is imperative to utilize some form of congestion control to avoid link congestion and to provide overall fairness in the Internet, the congestion control itself should be varied according to the needs of the application. It is important to note that all DASH client algorithms inherently implement some form of congestion control and thus, in current systems there are two control loops present; one at the transport layer for TCP and one in the DASH client. QUIC supports the use of both custom and standard congestion control protocols such as TCP Cubic, TCP NewReno and the state-of-the art bottleneck bandwidth and round-trip propagation time (BBR) based congestion control [5]. ABR clients can disable congestion control in the QUIC transport layer and continue to use their existing

congestion control while QUIC provides the benefits of pacing and assistance for loss recovery in the form of NACKs. Legacy implementations for application layer congestion control include protocols such as Real-time Transport Protocol (RTP) [8].

6 RELATED WORK

Recent works that evaluate alternative transport protocols for adaptive bitrate (ABR) video streaming include the work by McQuistin et al. [14]. They present *TCP Hollywood*, a TCP variant, which implements out-of-order delivery and inconsistent retransmissions in order to improve good-put of video streaming applications. Timmerer et al. [22] present a performance evaluation of 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 [21], a demonstration by Szab et al. provides a new congestion control mechanism using QUIC that aggressively varies download rate according to a buffer-based priority level assigned by the ABR streaming client. Carclucci et al. [6] present results that compare TCP and QUIC under varying network conditions and buffer size. However, their work is restricted to the performance impact of QUIC for web applications in general whereas we specifically focus on DASH clients.

Legacy protocols that perform adaptive bitrate video streaming over UDP include systems such as Real-time Transport Protocol (RTP) [8] and Stream Control Transport Protocol (SCTP) [18]. Similar to QUIC, SCTP also allows multiplexing of multiple chunks 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 Video Transport Protocol (VTP) which is designed and evaluated by Balk et al. [4]. 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.

7 CONCLUSION

In this paper, we investigated the performance of adaptive bitrate streaming algorithms over QUIC. Specifically, we studied the QoE performance of different DASH quality adaptation algorithms that are either solely based on the playout buffer filling, or on the video segment download rate, or on both. We investigated different QoE performance metrics such as the average quality bitrate, a measure of quality variations denoted spectrum and the average video stalling duration. Here, we evaluated the performance of DASH quality adaptation algorithms on top of QUIC and TCP in a controlled testbed environment, as well as, in the wild. Our (preliminary) evaluation results show that using the unmodified DASH algorithms on top of QUIC does not provide the anticipated QoE performance boost when compared to the standard DASH over TCP setup. Although we observe a lower magnitude of quality variations, the degradation of streamed quality bitrate with the use of QUIC is detrimental to overall QoE.

REFERENCES

[1] NetFlix. http://www.netflix.com. Accessed: 2017-03-10.
[2] YouTube. https://www.youtube.com. Accessed: 2017-03-10.
[3] Caddy web server. https://caddyserver.com/, Accessed 3-9-2017.
[4] A. Balk, M. Gerla, M. Sanadidi, and D. Maggiorini. Adaptive mpeg-4 video streaming with bandwidth estimation: Journal version. *vol*, 44:415–439, 2003.
[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. Bbr: Congestion-based congestion control. *Queue*, 14(5):50, 2016.
[6] 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.
[7] Y. C. Chen, D. Towsley, and R. Khalili. Msplayer: Multi-source and multi-path video streaming. *IEEE Journal on Selected Areas in Communications*, 34(8):2198–2206, Aug 2016.
[8] R. F. H. Schulzrinne, S. Casner and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications, 2017.
[9] 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.
[10] 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.
[11] T. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proc. of ACM SIGCOMM*, pages 187–198, 2014.
[12] 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.
[13] S. Lederer, C. Müller, and C. Timmerer. Dynamic adaptive streaming over HTTP dataset. In *ACM MMSys*, pages 89–94, 2012.
[14] S. McQuistin, C. Perkins, and M. Fayed. Tcp hollywood: An unordered, time-lined, tcp for networked multimedia applications. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 422–430, May 2016.
[15] R. Ricci, E. Eide, and The CloudLab Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *USENIX ;login:*, 39(6), Dec. 2014.
[16] J. Roskind. Quic (quick udp internet connections), 2017.
[17] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *Proc. of IEEE INFOCOM*, pages 1–9, April 2016.
[18] R. Stewart, Q. Xie, K. Morneault, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol, 2017.
[19] T. Stockhammer. Dynamic adaptive streaming over HTTP–: standards and design principles. In *ACM MMsys*, pages 133–144. ACM, 2011.
[20] I. Swett and J. Iyengar. QUIC loss recovery and congestion control, 2017.
[21] 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.
[22] C. Timmerer and A. Bertoni. Advanced transport options for the dynamic adaptive streaming over http. *arXiv preprint arXiv:1606.00264*, 2016.
[23] S. I. ULC. Global Internet phenomena report 2016, 2016.
[24] 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.
[25] M. Zink, J. Schmitt, and R. Steinmetz. Layer-encoded video in scalable adaptive streaming. *IEEE Transactions on Multimedia*, 7(1):75–84, Feb 2005.