Public Review for

Playing Chunk-Transferred DASH Segments at Low Latency with QLive

Praveen Kumar Yadav, Abdelhak Bentaleb, May Lim, Junyi Huang, Wei Tsang Ooi, Roger Zimmermann

In this well-written paper, the authors present a thorough overview of QLive, a live video streaming system using DASH, that aims to reduce transport latency. It relies on chunked transfer encoding (CTE) with the common media application format (CMAF) to start delivering chunks of each segment before the segment is fully available in real-time. Its main contribution lies in a set of techniques to resolve fundamental problems on bandwidth estimation (e.g. present in the LoL algorithm it is based on), latency control and bitrate selection.

QLive first addresses application-level bandwidth estimation through the use of an existing chunk parser and a chunk data filter and then models the rate and playback speed adaptation as an $M/D/1/K$ queue to control the latency within a given limit. In this way, QLive effectively optimizes the interplay between playback speed and other system parameters such as the estimated bandwidth, buffer latency and representation bitrate. A more clear justification of why the approximation of the rate adaptation component by an $M/D/1/K$ queue is appropriate for the envisioned scenario could have been beneficial.

QLive has been implemented on the Dash.js reference player and evaluated for various performance metrics, both on a local proof-of-concept setup, against three other algorithms, and over the Internet, for a diverse and realistic set of network and system scenarios. The authors have then assessed the aggregated Quality of Experience (QoE) of QLive, using two state-of-the-art metrics, showing improved bandwidth estimation accuracy and latency control with minimal changes in playback speed. While the authors provide separate results for the different network profiles, the detailed impact of each parameter on the QoE is still missing. The opportunity to present a subjective assessment of QLive has not been taken. Especially to also assess the impact of variable playback speed, currently missing in the state of the art, this would have been helpful.

Public review written by
Tim Wauters
Ghent University – imec,
Ghent, Belgium

Playing Chunk-Transferred DASH Segments at Low Latency with QLive

Praveen Kumar Yadav*
Atlastream
praveenkyadav@u.nus.edu

Abdelhak Bentaleb
School of Computing
National University of Singapore
bentaleb@comp.nus.edu.sg

May Lim
School of Computing
National University of Singapore
maylim@comp.nus.edu.sg

Junyi Huang*
School of Software Engineering
Xi'An Jiaotong University
huangjunyi1112@163.com

Wei Tsang Ooi
School of Computing
National University of Singapore
ooiwt@comp.nus.edu.sg

Roger Zimmermann
School of Computing
National University of Singapore
rogerz@comp.nus.edu.sg

ABSTRACT

More users have a growing interest in low latency over-the-top (OTT) applications such as online video gaming, video chat, online casino, sports betting, and live auctions. OTT applications face challenges in delivering low latency live streams using Dynamic Adaptive Streaming over HTTP (DASH) due to large playback buffer and video segment duration. A potential solution to this issue is the use of HTTP chunked transfer encoding (CTE) with the common media application format (CMAF). This combination allows the delivery of each segment in several chunks to the client, starting before the segment is fully available in real-time. However, CTE and CMAF alone are not sufficient as they do not address other limitations and challenges at the client-side, including inaccurate bandwidth measurement, latency control, and bitrate selection.

In this paper, we leverage a simple and intuitive method to resolve the fundamental problem of bandwidth estimation for low latency live streaming through the use of a hybrid of an existing chunk parser and proposed filtering of downloaded chunk data. Next, we model the playback buffer as a $M/D/1/K$ queue to limit the playback delay. The combination of these techniques is collectively called QLive. QLive uses the relationship between the estimated bandwidth, total buffer capacity, instantaneous playback speed, and buffer occupancy to decide the playback speed and the bitrate of the representation to download. We evaluated QLive under a diverse set of scenarios and found that it controls the latency to meet the given latency requirement, with an average latency up to 21 times lower than the compared methods. The average playback speed of QLive ranges between 1.01 - 1.26× and it plays back at 1× speed up to 97% longer than the compared algorithms, without sacrificing the quality of the video. Moreover, the proposed bandwidth estimator has a 94% accuracy and is unaffected by a spike in instantaneous playback latency, unlike the compared state-of-the-art counterparts.

*This work was done when the author was at National University of Singapore.

MMSys 21, September 28-October 1, 2021, Istanbul, Turkey

ACM ISBN 978-1-4503-8434-6/21/09.
https://doi.org/10.1145/3458305.3463376

CCS CONCEPTS

• **Information systems → Multimedia streaming**.

KEYWORDS

DASH, live streaming, low latency, queuing model, CMAF, CTE

ACM Reference Format:
Praveen Kumar Yadav, Abdelhak Bentaleb, May Lim, Junyi Huang, Wei Tsang Ooi, and Roger Zimmermann. 2021. Playing Chunk-Transferred DASH Segments at Low Latency with QLive. In *12th ACM Multimedia Systems Conference (MMSys '21) (MMSys 21), September 28-October 1, 2021, Istanbul, Turkey*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3458305.3463376

1 INTRODUCTION

Video streaming represents a significant portion of Internet traffic today. The popularity of live video streaming applications, such as online video games, live event streaming, and virtual reality applications, is increasing with an expected growth of fifteen-fold to reach 17% of the total traffic by the end of year 2022 [2]. These applications require low end-to-end latency (also called glass-to-glass latency), i.e., the time lag between video capture and the actual playback time at the client, to enable real-time interaction while providing high quality and avoiding rebuffering events. Dynamic adaptive streaming over HTTP (DASH) [4] clients were initially designed for video-on-demand (VoD) services without any stringent latency requirements in mind. Therefore, legacy DASH solutions face a severe problem in delivering low-latency live streaming. The currently achieved latency by these solutions ranges from 6 to 40 seconds as shown in prior studies [5, 6, 55]. This unacceptable latency is often due to a large playback buffer and segment duration. In traditional DASH solutions, the origin server has to wait for an entire segment to be encoded and packaged before being pushed to a content delivery network (CDN). This video contribution process requires at least one segment duration delay. In practice, the DASH video player has to buffer several segments (e.g., three segments in Apple HTTP live streaming) to start decoding and rendering. These serial delays, from video capture until rendering, can significantly increase the end-to-end latency.

One way to reduce the end-to-end latency is to use a shorter segment duration (one second or less). Although this solution can control the latency to within target limits to some extent, it has several problems: (*i*) decrease in encoding efficiency, (*ii*) frequent

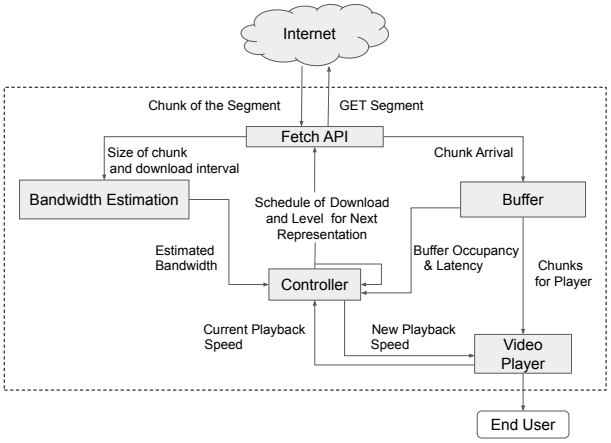


Figure 1: Generic DASH rate and speed adaptation process for low latency streaming using CTE and CMAF.

quality changes (quality instability), and (*iii*) tremendous increase in the number of HTTP requests and responses. To avoid these problems while keeping the latency small, chunked transfer encoding (CTE) with MPEG Common Media Application Format (CMAF; ISO/IEC 23000-19) [23] has recently emerged as the standard packager for low-latency delivery. CTE is one of the main features of HTTP/1.1 (RFC 7230) [20] that allows delivery of a segment in small pieces called chunks. A chunk can be as small as a single frame, so that it can be delivered to the client near real-time, even before the segment is fully encoded and available at the origin side. In CMAF, a chunk is the smallest referenceable unit of a segment that contains Movie Fragment Box ("moof") and Media Data Box ("mdat") atoms. These atoms make a chunk independently decodable, although representation switching still happens at the first chunk of the segment that contains the Instantaneous Decoder Refresh (IDR) frame.

Although CTE and CMAF solution is a step forward in low latency streaming, it alone is not enough for reducing latency. It requires efficient content delivery networks (CDNs), resource-efficient encoders, and optimized adaptive rate and playback speed adaptation algorithms. Figure 1 shows an overall architecture of the DASH client for low latency live video streaming using CTE and CMAF. A DASH client first requests the video segment based on the information given in the media presentation description (MPD) file. The properties available in MPD, namely *availabilityStartTime* and *publishTime*, allow the DASH client to calculate the instantaneous latency and request the latest available segment from the server. The Fetch API[1] (currently supported by major browsers) at the client enables the reading of a chunk out of a partially downloaded segment and push it to the playback buffer of the video player, from where the video player renders it at a specific speed decided by the controller. The controller is also responsible for estimating the bandwidth using the downloaded chunk's size (in bytes) and the time interval for its download. The controller implements an adaptive bitrate (ABR) algorithm that considers different heuristics based on the estimated bandwidth value and the buffer information, the required latency limits, and the instantaneous playback speed to decide the subsequent representation of the segment to download and the playback speed. This ABR algorithm is customizable as per the DASH client's requirements.

Unlike in the case of VoD streaming with DASH, the measured value of the throughput for the downloaded segment is not a reasonable estimate of the network transmission capacity [5, 34]; therefore, we use the estimated bandwidth instead. As explained in the previous studies, the throughput measurement problem using the ratio of segment size and segment download time always produces a value equal to (or slightly smaller than) the segment encoding bitrate due to inter-chunk idle periods introduced during CTE delivery [5, 6]. Hence, the ABR algorithm experiences incorrect throughput measurements, which prevents it from switching to higher bitrate levels. Moreover, a DASH client faces a longer latency if its buffer contains video chunks for a longer duration or a higher network latency due to a sudden drop in bandwidth. When a client plays the entire video at normal playback speed without skipping any of the chunks, the latency becomes cumulative. Therefore, its future value cannot go lower than the instantaneous value. One way to reduce the latency is to skip some segments and download the latest segment. However, this technique is not suitable for many applications such as online gaming and live sports streaming as the user would not like to skip key moments in a game to reduce the latency. Another technique for reducing the latency is speeding up the playback of the buffered chunks to play the later segments earlier. In such a case, the controller needs to decide a reasonable value for the playback speed for this latency control process, as an overly aggressive playback speedup leads to an empty buffer, causing a playback stall.

In this paper, we address the issues mentioned above and propose an $M/D/1/K$ queuing system-based model (termed *QLive*) for bitrate adaptation and playback speed control using the estimated bandwidth, buffer occupancy, instantaneous playback speed, and the total buffer capacity. We leverage a hybrid bandwidth estimation algorithm that combines the latest version of LoL's [38] bandwidth measurement module with a new chunk filtering mechanism using the *maximum transfer unit* (MTU).

We implement QLive over the latest `Dash.js` reference player [25] and compare its performance with recently published ABR algorithms[2] for low latency streaming that includes LoL [38], L2A [32], and Stallion [27].

Our method achieves 94% accuracy in the bandwidth prediction regardless of the instantaneous latency. Furthermore, the bitrate and playback speed adaptation algorithm controls the average latency to meet the target limit, with an average playback speed of 1.01 to 1.26 for different latency limits. Furthermore, the duration for playing at normal speed is up to 97% longer than the current state-of-the-art algorithms.

Paper Contributions. First, we leverage a simple and intuitive hybrid bandwidth estimation method [38] with high accuracy regardless of the instantaneous latency and network conditions. Next, we model the DASH client as $M/D/1/K$ queue to develop a joint bitrate and playback speed adaptation algorithm that avoids excessive playback speed, keeps the latency within the target limits, and maintains a high quality-of-experience (QoE). Here, we measured

[1] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

[2] Note that LoL and L2A are integrated with `Dash.js` player (v3.2.0) as default ABR algorithms for low latency streaming.

the QoE using two state-of-the-art QoE modeling formulas for adaptive video streaming [3, 62]. We evaluated QLive extensively using different network traces with different segment durations for different latency thresholds. We also tested QLive on the public Internet to stream video across two continents during the current Covid-19 pandemic when the increase in video traffic is burdening the Internet. QLive controls the latency without causing excessive playback speedup while avoiding stalls. We also show the effects of the segment duration on the instantaneous and average latency.

Paper Organization. Section 2 provides an overview of the related work to improve live streaming performance. Section 3 presents the hybrid bandwidth estimation method. Section 4 describes the proposed $M/D/1/K$ queuing model for low latency live streaming DASH clients as well as the QLive algorithm. Section 5 presents the evaluation of QLive. We conclude in Section 6.

2 RELATED WORK

We now discuss the existing approaches to improve live streaming.

Encoding optimization. Retransmission of lost packets adds significant latency for progressive video streaming applications. Liang et al. [37] have proposed an optimal picture selection technique for layered encoding based on video content, packet loss probability, and channel feedback to avoid packet retransmission for reducing the latency. Varodayan et al. [58] have proposed a feedback mechanism to detect errors based on picture comparison in time and space to improve the visual quality in live multicast adaptive video streaming. In another approach, Zhou et al. [67] have proposed a bitrate independent encoding to reduce the latency in low-throughput networks using the number of encoded symbols. In live streaming, the encoding of a segment cannot finish before all its frames have been produced (i.e., received from the camera). Therefore, a client gets an error code if it requests a segment earlier than its availability, which leads to re-requesting the segment. On the other hand, another approach lets the server wait to complete the segment before responding to the client. In both scenarios, the segment duration has a significant impact on latency. Swaminathan et al. [55] have proposed dividing a segment into equal-sized chunks to reduce this latency dependence on the segment duration. Thus, the server responds with the available chunks upon receiving the request without completing the segment encoding. To further mitigate the latency dependency on encoding time, Bouzakaria et al. [8] have proposed a frame-level transmission. The work shows that a frame-level transmission is possible with acceptable overhead. In 2017, MPEG standardized the chunk-based encoding with CMAF. Yahia et al. proposed a selective frame dropping using HTTP/2 to reduce the latency while maintaining visual quality [61]. Heikkinen et al. [28] and Aparicio et al. [1] have addressed resource optimizations, whereas Bentaleb et al. [6] have analyzed the impact of encoding parameters on live streaming.

Transmission optimization. Peer-to-peer streaming is a well-studied mechanism for live video streaming. Various algorithms have been proposed for the admission control and identification of peers [11, 59]. Shen et al. proposed that peers available in a LAN network can support each other [53], whereas Sweha et al. proposed the inclusion of angel servers to support peer-to-peer streaming [56]. Most of these approaches are less efficient than using CDNs, as the availability of a peer is not guaranteed, and many do not support HTTP-based live streaming. Roverso et al. have proposed a peer-to-peer-based approach that supports HTTP-based live streaming and has comparable performance to CDNs by using proactive-reactive hybrid prefetching [52]. Bruneau et al. have proposed a mechanism to download chunks both from multiple peers and dedicated servers [10]. In contrast, Evensen et al. [17] proposed using 3G and WiFi to improve the quality of live streams. Mirshokraie et al. optimized video encoding using layered encoding for peer-to-peer streaming [40]. Other transmission optimization techniques to improve live streaming include the use of HTTP/2 to reduce the number of requests [31, 45], a hybrid of broadband and multicast [36], optimization of TCP congestion control [7], multi-server streaming [12], scheduling users to different CDNs [65], and selective retransmission of frames [50].

Player optimization. Due to the higher cost-effectiveness of HTTP-based streaming, Akamai proposed a feedback control adaptive streaming for live video [14], where the server throttles the sending rate based on the network conditions and the client buffer. De Cicco et al. [15] extended this approach to mitigate the network underutilization in Akamai's approach by matching the video bitrate to the network throughput. Steinbach et al. [54] showed a reduction of latency by increasing the playback speed through simulation. Gualdi et al. [26] have proposed continuous playback speed modifications for reducing the latency, whereas LAPAS [64] used speed adaptation to reduce the latency for non-CMAF streams by matching the playback finishing time of a segment with the availability of the next segment. However, such approaches can hamper the user experience as the end-user would rarely get to watch the video at normal speed. Hooft et al. [57] have proposed prefetching of a few segments along with the MPD file using HTTP/2 to reduce the latency. As mentioned in the earlier encoding optimization paragraph, the latency for non-CMAF encoding is significantly dependent on the segment duration. Therefore, reducing the latency for such encoded videos is very difficult as the client can only play the segments after fully downloading them. To handle this situation, LAPAS controls the playback speed of a segment to match the playback finishing time of the segment with the next segment's availability. Therefore, the algorithm reduces the playback speed if the latency falls below the latency limit and increases the playback speed if it is more than the latency limit. In such a scenario, a latency limit that is too low leads to a playback stall, whereas maintaining the buffer increases the latency, which makes CMAF an obvious choice over other formats for low latency streaming.

Vabis [19] proposed a server-side bitrate adaptation using reinforcement learning to optimize the QoE. In the context of low-latency with CTE and CMAF, Lim et al. [38] have re-visited and extended several vital components (collectively called Low-on-Latency, LoL) in adaptive streaming systems. LoL includes three essential modules: bitrate adaptation (both heuristic and learning-based), playback control, and throughput measurement. Karagkioules et al. [32] designed the Learn2Adapt (L2A) ABR algorithm based on online learning and online convex optimization. Similarly, Gutterman [27] developed Stallion, a simple algorithm that uses a sliding window algorithm to measure the mean and standard deviation of both the bandwidth and latency to perform ABR decisions. More

algorithms proposed bitrate adaptations along with skipping of frames for controlling the latency [30, 43, 44, 66]. Montagud et al. [41] and Rainer et al. [47, 48] have proposed playback speed adjustment algorithms for media synchronization and preventing buffer underflow and overflow in adaptive media playout regardless of latency.

In the proposed work, we focus on controlling the latency for video playback in a DASH client without skipping frames using limited playback speed modifications to improve QoE while maximizing the normal playback speed duration.

3 BANDWIDTH ESTIMATION

We now describe the need for bandwidth estimation and describe a method to get a more accurate estimate regardless of the instantaneous latency and network conditions.

The encoding time of a segment in live video streaming is equal to its content duration since the encoder has to wait to receive every frame from the camera in real-time. This encoding time affects the throughput measurement at the client end [5, 35]. For example, a segment encoded at 2000 Kbps with a segment duration of 4 seconds has a total size of 8000 Kb and takes 4 seconds to generate entirely at the server (origin). In the hypothetical case of zero network delay, the segment requested at the beginning of its encoding at the server needs 4 seconds to get fully downloaded at the client's end, resulting in a throughput measurement of 2000 Kbps. Therefore, in the case of additional network latency, the measured throughput cannot exceed the segment's bitrate value and cannot depict the available bandwidth. This inaccuracy in measuring the bandwidth calls for an alternative measurement for bandwidth estimation. In this paper, we leverage a hybrid bandwidth estimation algorithm that combines the latest version of LoL's [38] bandwidth measurement module with a new chunk filtering mechanism using the *maximum transfer unit* (MTU). The hybrid algorithm (LoL bandwidth measurement module + MTU filtering) works in three stages:

(1) *Chunk Parsing*: The support for the Fetch API in web browsers provides an interface for fetching resources on the Internet with more flexibility than the earlier XMLHttpRequest approach. The API allows the reading of the response body content in real-time before the completion of its download. A DASH client reads part of the HTTP response body by using the *read()* method of Fetch API, creates a copy of it, and pushes it to the playback buffer. This functionality enables the playing of a chunk before downloading the entire segment. Our algorithm leverages this capability of the Fetch API, tracks the chunk's download progress, and parses the chunk payload in real-time to check if "moof" and "mdat" boxes (or atoms) in fragmented MP4 data are present. It identifies the exact arrival of the beginning and ending times of each chunk download. When the filter captures a "moof" box of a chunk, the algorithm stores the timestamp as the beginning time of the chunk download using *performance.now()*. The next step stores the ending time of the chunk and its size (in bytes) when the chunk is fully downloaded (end of "mdat").
(2) *Chunk Filtering*: Once the client downloads a segment, our algorithm triggers a two-phase filtering process to remove the inter-chunk idle periods and noise. First, it checks again for the time intervals for which "moof" is present, and the bytes received are equal to or more than the MTU. If the condition is satisfied, it considers this chunk in the bandwidth calculation. Otherwise, the client considers the interval as an idle period and filter out the chunks from the bandwidth estimation.
(3) *Segment Bandwidth Estimation*: Our algorithm uses the ratio of the total bytes and their download time for such intervals as the estimated bandwidth (see Equation 1). The MTU value used for filtering by the DASH client is 1448 bytes because the bytes allocated for TCP and IP headers and timestamps are not available at the application layer.

$$
\begin{aligned}
&For\ each\ downloaded\ chunk\ j\ of\ segment\ i: \\
&\quad Total\ Size_i = Total\ Size_i + Size_{chunk\ j} \\
&\quad Total\ Time_i = Total\ Time_i + Download\ Time_{chunk\ j} \\
&\textbf{Subject to}: \\
&\quad Size_{chunk\ j} \geq MTU \\
&\quad Flag_{moof} = TRUE \\
&Estimated\ Bandwidth_i = Total\ Size_i / Total\ Time_i
\end{aligned}
\tag{1}
$$

We observed that LoL's bandwidth measurement module successfully detects and eliminates idle periods between chunks introduced during CTE delivery. However, we found that in some particular cases where the instantaneous latency drops below one second, the accuracy of the throughput measurement drops (as shown in Section 5.5). The filtering technique considers some chunks of a segment as an idle period and filters them out from the bandwidth calculation. To mitigate the issue, we added another filtering approach based on MTU; thus, we can maintain high estimation accuracy. The intuition behind the method is as follows: (*i*) The presence of "moof" atoms from the portion of the response body implies the beginning of a new chunk. (*ii*) Bytes received by the client are more than or equal to the MTU implies that the fragmentation is performed for the transmission because the available packet is too large for transmission over the network interface. Therefore, the chunk's download is lesser affected by the encoding latency. We show that the method has consistent accuracy for different segment durations and network conditions in Section 5.5.

4 QLIVE SYSTEM MODEL

With a mechanism to accurately estimate the available bandwidth, we now describe how QLive performs joint bitrate and playback speed adaptation. QLive approximates the playback buffer as a queuing system.

Using a queueing model, QLive estimates the DASH client's buffer occupancy convergence for a given bitrate while playing the video at a specific rate with the given network bandwidth. QLive then selects the representation of the next segment to download and the playback speed of the video so that the buffer occupancy converges to the ideal value.

A DASH client can be approximated as an $M/D/1/K$ queuing system where the live video stream chunks that belong to a fixed duration segment arrive in the queue. Here, chunks' arrival in the queue with capacity **K** is assumed to be **M**emoryless, and single decoder (**1**) process or decode the chunks of a segment at a

Deterministic rate. This model has been shown to approximate the DASH client playback buffer well [60].

Each segment consists of several chunks. Let each segment of the video is encoded into $\mathcal{L}$ representation levels, with bitrate values $R = \{r_1, r_2, ..., r_{\mathcal{L}}\}$, and $r_i < r_j$ if $i < j$. The DASH client starts playing the video as soon as the chunk arrives in the queue. Although the video is played when a chunk is available, the decision to switch the video's representation level is taken by the client at the segment boundary where IDR frames are present. The client requests a new segment once it finishes the download of the previous one. The client model is similar to QUETRA [60] for VoD streaming, but some changes to suit low latency streaming are required.

We now describe why QUETRA's model is not applicable for low latency live streaming. The model calculates the arrival rate of the segment as $\lambda = b/(d \times r)$, where b is the measured throughput for the entire segment download, d is the segment duration, and r is the bitrate of the segment. The service rate is calculated based on the video segment's duration played per second, which is $\mu = s/d$. Here, s is the playback speed. The queuing utilization $\rho = \lambda/\mu$ is therefore $b/(r \times s)$. For simplicity, the model considers $s = 1$, which is the common scenario for VoD streaming, where the user plays back the video at the normal speed. Therefore, the utilization is simplified to $\rho = b/r$. Furthermore, the model considers the buffer capacity K as the duration of the video available in the buffer (in seconds). Using the analysis for finite buffer queue by Brun et al. [9], QUETRA calculates the average buffer occupancy X for a given ρ and K. The average buffer slack BS is the difference between buffer capacity K and the average buffer occupancy X. In ideal case where the segment bitrate r equals to the throughput b, gives 100% utilization ($\rho = 1$). When $\rho = 1$, the BS can be approximated with $K/2$. By choosing the bitrate such that the buffer occupancy B_t at the time t is as close to the buffer slack as possible, the model, in the hypothetical case where the bitrate values are continuous, converges to the buffer occupancy to $K/2$. Therefore, the DASH bitrate adaptation algorithm determines the bitrate of the representation to download for the next download step i with an estimated throughput b_i as:

$$r_i = \arg\min_{r \in R} |BS_{K,r,b_i} - B_t| \tag{2}$$

The slack calculated in the above equation uses a predetermined playback speed of 1, which is the general case for VoD streaming, where the user manually selects the playback speed. However, in the case of low latency live streaming, the client needs to determine the representation level with a given bitrate and the playback speed when the instantaneous latency is beyond the pre-specified limit. Also, Equation 2 aggressively choose a bitrate higher than the capacity if the slack caused by it is nearer to the ideal value. Therefore, such selection can cause rebuffering for low latency streaming, especially when the latency limit is very small, e.g., 1 second.

4.1 Proposed Model for Low Latency

CMAF-based video segmentation and the Fetch API allow chunks as small as a single frame to be added to the buffer and decoded before downloading the entire segment. Therefore, we measure the buffer occupancy as the number of frames available in the buffer for playing. The algorithm calculates the value of buffer occupancy by taking the product of buffer occupancy in seconds and the video's frame rate. Having a higher buffer occupancy increases the latency as the chunk waits for a longer time in the buffer to get played. So we set the buffer capacity K to the required latency limit. The algorithm first finds the ideal value of ρ for keeping the buffer half-filled by considering the instantaneous buffer occupancy as the expected average buffer slack as per the etiquette of $M/D/1/K$ queue. Using this value of ρ and the estimated bandwidth BW by the proposed method in Section 3, we calculate the product of bitrate and playback speed with the help of queuing utilization $\rho = \lambda/\mu$ as:

$$r \times s = BW/\rho \tag{3}$$

4.2 QLive Design

Here we describe the rate and speed adaptation under different scenarios using the Equation 3 derived in the previous section.

Scenario 1: Latency is within the required limit. The set of available bitrate values R is discrete in practice. Therefore, in a case when the latency is within the required latency limit, QLive selects the playback speed $s = 1$ and the representation level with the highest bitrate that is less than or equal to $r^* = BW/\rho$, based on Equation 3. Let this selected representation level be y with the bitrate r_y. The algorithm uses the values y, r_y, and r^* for rate and speed adaptation in the next scenarios.

Scenario 2: Latency is more than the required limit and the buffer occupancy is high. When the bandwidth increases abruptly, a client receives many chunks in the buffer subject to the server's chunk availability. For instance, suppose a client plays a live stream with segments of 1 second, and the instantaneous latency is 20 seconds due to a lower bandwidth in the past. This value implies that 20 segments are available at the server for download. When the bandwidth increases sharply, the buffer occupancy will also increase abruptly. Suppose our latency limit is 1 second, and the client downloads 3 segments in less than 1 second due to a sudden increase in bandwidth. In that case, the buffer occupancy will grow more than the latency limit, whereas the instantaneous latency will still be around 17 seconds. In such a situation, QLive selects an adjacent lower representation than the representation it would select for $s = 1$, which is $y-1$ with the bitrate r_{y-1}. This selection of representation level allows the algorithm to select a playback speed $s > 1$, which satisfies the condition $r_{y-1} \times s = BW/\rho$. It is important to note that while calculating ρ using buffer occupancy at the time t, QLive uses the effective instantaneous buffer occupancy B_t/s instead of B_t because $s > 1$, which leads to faster drainage of the buffer. Such a calculation of ρ accounts for the change in average buffer occupancy due to the change in playback speed to avoid rebuffering stalls. Since ρ is a monotonically decreasing function of instantaneous buffer occupancy, and there are a finite set of values for s due to the web browser and QoE constraints, we can approximate the value of s that satisfies the relationship between r, s, BW, and ρ.

Scenario 3: Latency is more than the required limit with lower buffer occupancy, causing feasible bitrate to be lower than the lowest available bitrate. When the network bandwidth is lower than the bitrate for the selected representation level, it

causes the latency at the client-side to increase when the buffer occupancy decreases. The buffer occupancy is very low in such a case, whereas the latency is beyond the latency limit. This lower value of the buffer occupancy leads to a r^* value lower than the bitrate of the lowest available representation. Therefore, QLive selects the lowest representation level with bitrate r_1 and the playback speed $s < 1$, which satisfies the condition $r_1 \times s = BW/\rho$. Here, QLive also uses B_t/s to calculate ρ because the buffer is now draining slowly due to slower playback speed. Since the buffer occupancy is measured in the unit of time in the model, draining the buffer slower effectively increases the buffer occupancy.

5 EVALUATION

To evaluate QLive, we compare it against three algorithms designed for low-latency scenarios.

The first one is LoL [38] that develops three main modules: a self-organizing map (SOM) learning-based ABR algorithm, playback speed control, and throughput measurement. The ABR algorithm of LoL considers four heuristics as an input (measured throughput, latency, current buffer level, and QoE [3]) in the SOM model to perform an ABR decisions. It also develops a robust throughput measurement algorithm that tracks when a chunk arrives and its download finishes.

The next algorithm is L2A [32], which uses an online optimization convex optimization framework to formulate ABR selection problem in low-latency live streaming and propose an online learning rule to solve the optimization problem. The main intuition behind L2A is to learn the best policy that can select a suitable bitrate for each segment downloaded. It does so without requiring any parameter tuning, modifications according to the application type, statistical assumptions for the channel, or bandwidth estimation.

The third algorithm, Stallion [27], uses the throughput-based ABR of `Dash.js` with the slight modification that incorporates a sliding window technique to measure the mean and standard deviation of both the throughput and latency and then performs ABR decisions. This modification makes Stallion react well to low latency requirements.

The playback speed algorithm for LoL, L2A, and Stallion is based on the default `Dash.js` algorithm and is independent of the bitrate adaptation algorithm. While L2A and Stallion use the original playback speed algorithm, LoL modifies it to consider the buffer state before changing the playback speed to avoid rebufferings. For a fair comparison, we use the default playback speed algorithm of `Dash.js` for LoL, L2A, and Stallion. The algorithm controls the latency by increasing the playback speed based on the latency limit and the instantaneous latency. The algorithm first calculates the ΔL, which is the difference between the instantaneous latency and the latency limit. This value is used for the calculation of playback speed as $((2 \times s_{max})/(1 + e^{-\Delta L \times 5})) + 1 - s_{max}$. Here, s_{max} is the upper threshold for the playback speed. There are two main issues with this playback speed control algorithm. First, it decreases the playback speed when the latency is below the limit. Second, if the latency is higher than the limit and the buffer is empty, the algorithm still increases the playback speed to reduce the latency, which can cause an increase in the playback stalls. Besides, we used the hybrid bandwidth estimation technique (Section 3) for all ABR algorithms, including QLive.

5.1 Video Sample

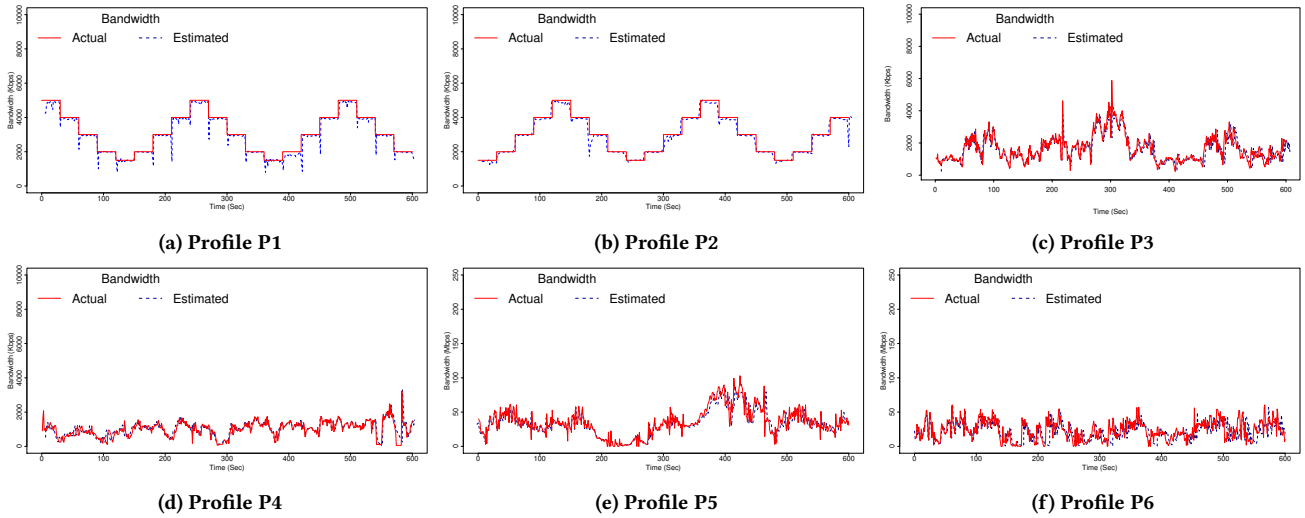
We use the software library provided by Streamline [29] that generates a synthetic CMAF-based live stream with continuous motion and different colors (random pattern). It uses FFmpeg (https://ffmpeg.org/) to encode and package the streams in CMAF format and a Python origin server to deliver the packaged chunks to the client using CTE. The original version of the source code generates the video stream at one bitrate. We modified the code to generate the video stream with three bitrates, chunked at the frame level. The highest bitrate for live streaming suggested by Facebook is 4000 Kbps, although there is no guideline for the lowest value [18]. YouTube suggests the lowest bitrate in the range of 300 Kbps to 700 Kbps for live streaming [63]. There are various other bitrate recommendations in the similar range 500 Kbps to 4000 Kbps from other platforms [13, 21, 22]. Therefore, we used three values 500 Kbps, 2500 Kbps, and 4000 Kbps. In addition, we choose the segment duration that is less than or equal to the required latency limit. E.g., for a 3-second latency limit, we use segment duration of 1, 2, and 3 seconds. Similarly, for a 2-second latency limit, we use segment duration of 1 and 2 seconds, and for 1-second latency limit, we use just a segment duration of 1 second. We describe the effect of choosing a segment duration with a higher value than the latency limit in Section 5.5 by analyzing the algorithms with a 6-second segment and a 3-second latency limit. The different playback speeds of algorithms can cause a difference in segments played for a given time. Therefore we run each experiment for 10 minutes instead of accounting for the number of segments.

5.2 Network Profiles

We used six network profiles for evaluation. Table 1 shows the summary of all the profiles. Profiles P1 and P2 are taken from the DASH Industry Forum Guidelines [24]. It has five levels with rate {1500, 2000, 3000, 4000, 5000} Kbps, with corresponding delay of {100, 88, 75, 50, 38} ms, and packet loss {0.12, 0.09, 0.06, 0.08, 0.09}% varying at the interval of the 30 seconds. P1 follows a high-low-high pattern; P2 follows a low-high-low pattern. Profiles P3 and P4 are HSDPA network trace from a moving car and train with the rate ranging from 241 to 5876 Kbps and 3 to 3344 Kbps varying at the interval of 1 second [51]. These two profiles are very challenging as the bandwidth is very low, and there are a few periods where avoiding rebuffering is challenging. Profiles P5 and P6 are 4G/LTE network trace from a moving car and train with the rate ranging from 0 to 103033 Kbps and 0 to 60555 Kbps varying at the 1-second interval [39].

5.3 Evaluation Metrics

First, we evaluated the accuracy of our bandwidth estimation algorithm. Next, we evaluated the traditional performance metrics for bitrate adaptation, namely, average bitrate, number of changes in representation, the magnitude of changes in quality (as the difference in bitrate values), number of stalls, duration of each stall, latency, and playback speed. We used two QoE models (Yin et al. [62] and Twitch grand challenge [3]) to calculate the QoE using



(a) Profile P1 (b) Profile P2 (c) Profile P3

(d) Profile P4 (e) Profile P5 (f) Profile P6

Figure 2: Actual and estimated bandwidth for different network Profiles.

Network profile	Inter-variation duration (sec)	Min (kbps)	Max (kbps)	Average (kbps)	Standard Deviation (kbps)	Delays (ms)	Packet Loss %	Type
*P*1	30	1500	5000	3062.5	1208.23	38 to 100	0.06 to 0.12	High-Low-High
*P*2	30	1500	5000	3062.5	1208.23	38 to 100	0.06 to 0.12	Low-High-Low
*P*3	1	241	5876	1754	884.29	-	-	HSDPA Trace
*P*4	1	3	3344	667.58	569.74	-	-	HSDPA Trace
*P*5	1	0	103033	2161.49	19504.45	-	-	4G/LTE Trace
*P*6	1	0	60555	23072.71	13878.16	-	-	4G/LTE Trace

Table 1: Characteristics of Network Profiles.

these metrics. Finally, we evaluated the latency-dependent matrices. We measured the average latency to check if an algorithm can control the latency with the preset limits. The instantaneous value of latency is critical for applications such as online gaming and video surveillance. Therefore, we checked for the maximum instantaneous value of latency as well. We also checked the average playback speed and the duration for which video is played at the normal speed by different algorithms.

5.4 Implementation and Experimental Setup

We implemented QLive on `Dash.js` [25] video reference player (v3.1.3) Other low-latency ABR algorithms, LoL [38], L2A [32], and Stallion [27], are already integrated in `Dash.js`. We evaluated the algorithms mentioned above with three latency limits of 1, 2, and 3 seconds. For the playback speed adaptation, we used the YouTube recommendation speed limit, i.e., 0.25 times to 2 times of normal playback speed [46]. We used an Apache Web server to host the `Dash.js` video player containing the four bitrate adaptation algorithms for comparison. The FFmpeg encoder with CMAF packager and origin ran on the server provided by Streamline [29]. The server and client ran on two different Linux-based machines connected by a router. We used the *tc* NetEm network emulator to control the network bandwidth according to the network profiles.

We tested the algorithms for each network profile and latency limit with different segment durations for 10 minutes of the total live video session. We repeated the network profiles for network traces that are shorter than 10 minutes. We measured all the metrics in the interval of 30 milliseconds to cover the frame-level changes in the buffer because buffer occupancy is often low in the low latency streaming.

5.5 Results and Comparison

We now compare and describe the performance of the different algorithms. First, we describe the performance enhancement for our proposed bandwidth estimation accuracy. Then, we compare the performance of different bitrate adaptation and speed control algorithms using the metrics such as average bitrate, changes in representation level, number of stalls, stall duration, playback speed, latency, and QoE. Unless specified otherwise, we present each algorithm's results for different latency limits and the different network profiles by averaging the results for different segment durations.

5.5.1 Bandwidth estimation accuracy. As we discussed in Section 3, the accuracy of the bandwidth estimation algorithm used in LoL is high in general. However, the method fails when the latency drops below 1 second. This phenomenon is not common with the LoL, L2A, and Stallion because these methods use `Dash.js` default

playback speed algorithm, which tends to slow down the playback speed to increase the instantaneous latency to the given limit whenever the latency value drops below the limit. LoL's bandwidth estimation filter out the ideal periods between the chunks. Therefore the method works well when multiple chunks of a segment are available for transmission at the server, i.e., when the latency is more than 1 second.

However, when the total end-to-end latency is in the order of chunk duration, a server must wait for the encoder to finish the encoding of a chunk for its transmission. In such a scenario, we need additional filtering for the latency caused by a single chunk's encoding duration to estimate the bandwidth accurately. As explained in Section 3, when bytes received by the client is more than or equal to the MTU, it implies fragmentation is performed for the transmission because the available packet is too large for transmission over that interface. Therefore, the chunk's download is lesser affected by the encoding latency. The QLive speed adaptation is different and does not aim to increase the latency regardless of the instantaneous value. Figure 3 shows one such example where the LoL's bandwidth estimation fails when used together with the bitrate adaptation of QLive. Around 330 seconds, the QLive latency drops below 1 second, and the algorithm also selects the lowest representation having the bitrate 500 Kbps. Such representation and the lower instantaneous latency give a false bandwidth estimate, which is too lower than the actual value. Therefore, QLive keeps on selecting the lowest representation, and the instantaneous value of latency remains lower than 1 second as the actual bandwidth is much higher than the representation's bitrate. Therefore, the error in the bandwidth estimation continues.

To address this issue, we modified the chunk filtering approach based on Equation 1. This modification leads to a very high average bandwidth estimation accuracy of 94%, with the minimum value of 87% for different profiles, and segment duration, calculated using Dynamic Time Warping with Manhattan distance [42]. Some of the inaccuracy is due to the experiment artifact, i.e., the segment's download falls in the time window when the NetEm is changing the bandwidth, causing the averaging of the values at the client's end. However, as we have demonstrated in Figure 2 for one of the experiment traces, the difference between the actual and estimated bandwidth is minimal regardless of latency.

5.5.2 Stall duration and number of stalls. Figure 4 shows the XY plot for comparing the number of stalls and their duration for different algorithms under different latency limits and network profiles. The points that are closer to the lower-left corner yield better performance in both dimensions, i.e., the minimum number of stalls with a minimum duration of stalls. QLive is almost in the lower-left corner for all the cases. The stall duration is between 0 to 1.63 seconds, and the number of stalls is between 0 to 2 on average.

The most challenging scenarios are Profiles P3 and P4, where the bandwidth is extremely low for a total duration of 8 seconds and 20 seconds, respectively, causing more stalls than other profiles. The situation becomes more challenging for 1-second latency limits as all the algorithms try to keep the buffer within 1 second to control the latency within limits. Small buffer capacity is also challenging with Profiles P1 and P2 that have induced delays on the top of throttled bandwidth. Nevertheless, QLive performance remains better in these challenging scenarios. On average, Stallion performance is worst with Profiles P1 to P4 as it does not consider the buffer occupancy for bitrate adaptation. Stallion has up to 10 stalls with 9.3 seconds duration on average for the profiles and latency limit setting. Similarly, L2A also has up to 4.04 seconds of stalls. LoL has a shorter stall duration but causes multiple small stalls with P3 and P4 for 1-second latency limits.

The playback speed adaptation is also a vital factor for stalls. The default `Dash.js` playback speed adaptation used by the compared methods increases the playback speed to control the instantaneous

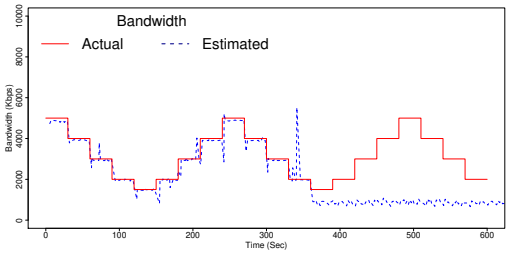


Figure 3: Bandwidth estimation error for algorithm used in LoL when latency drops below one second.

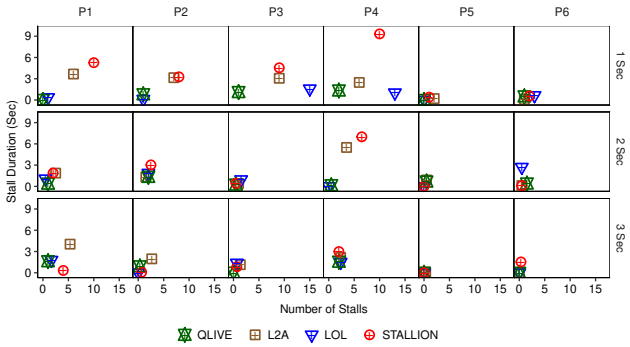


Figure 4: Stall vs. No. of stalls for different latency limits and network profiles.

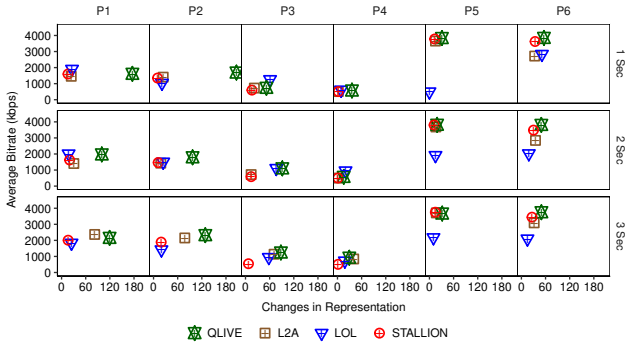


Figure 5: Bitrate vs. changes in representation for different latency limits and network profiles.

latency. In a scenario where latency is high during the stalls, an increase in the playback speed aids in increasing the stalls as the buffer occupancy shrinks faster.

5.5.3 Average bitrate and changes in representation. Figure 6 shows the XY plot for comparing the average bandwidth against the number of changes in representations for different algorithms under different latency limits and network profiles. The points that are closer to the lower-left corner yield better performance in both dimensions. We can see that QLive always has the highest bitrate – up to 35% higher than the compared methods. In contrast, LoL behaves conservatively to increase the bitrate to reduce the stalls. Therefore it has the lowest bitrate on average, but it also has fewer stalls than L2A and Stallion. Profiles P5 and P6 are the ones where bitrate is significantly different for different algorithms. LoL, L2A and Stallion have up to 86%, 29%, and 8.6 % lower bitrate than QLive, respectively, with Profiles P5 and P6.

Although the number of changes in the representations for QLive is highest, the values are less than 1/3 times the maximum possible values, based on the segment duration and total playback duration. Additionally, QLive also has the minimum stall duration as described in the previous section, and stall avoidance is more critical to QoE than changes in representation [16]. Looking deeper into the results, QLive has comparable changes in the representations with respect to other algorithms with Profiles P3 to P6. Only Profiles P1 and P2, having induced delays and packet losses, cause significant changes in representation levels for QLive.

Stallion has the closest bitrate value to the QLive for 1- and 2-second latency limit; On the other hand, it also has the highest stall duration on average. Similarly, L2A has the closest bitrate to the QLive for 3-second latency limit and has the highest stall duration. Therefore, it is vital to see the bitrate, changes in the representation levels, and stalls altogether.

5.5.4 Average and maximum instantaneous latency. Figures 6 and 7 show the average and the maximum latency at any instance for different algorithms under different latency limits. Applications like online video games, live auctioning, and video surveillance requires lower average latency and lower instantaneous latency, making both factors vital for performance evaluation.

QLive has the lowest average latency when compared to the other algorithms. Except for P3 with 3-second latency limit and P4, the average latency for QLive is always within limits. Its maximum latency at any instance is also the lowest. Other algorithms have variable performance for different latency limits. However, the pattern of their performance is similar for average latency and maximum latency. As we have seen for stalls, playback speed adaptation also affects the latency for these algorithms when the latency limit is small. When the default `Dash.js` playback speed adaptation used by the compared methods increases the playback speed to control the instantaneous latency, it causes playback stalls and further increases the latency. Therefore, L2A and Stallion have exceptionally bad performance with P4 with 1-second and 2-second latency limits. L2A has a maximum latency of 44 seconds and 20 seconds for when the latency limit is 1 second and 2 seconds respectively, whereas Stallion has maximum latency of 42 seconds and 21 seconds. LoL, being conservative for increasing bitrate, is not as affected for 1-second latency limit, but it has a maximum latency of 34 seconds for 2-second latency limits. The bandwidth for Profile P3 is also comparatively lower than Profiles P1, P2, P5, and P6 but is higher than the lowest bitrate most of the time. Therefore, the average latency with P3 is not as high as P4 for LoL, L2A, and Stallion and ranges between 3 seconds to 8.9 seconds. Profiles P1 and P2, having induced delay, also cause the latency to exceed the limits for LoL, L2A, and Stallion. Due to the high bandwidth in P5 and P6, all four algorithms have average latency closer to the limits. However, in P5 and P6, the bandwidth goes below the lowest available bitrate

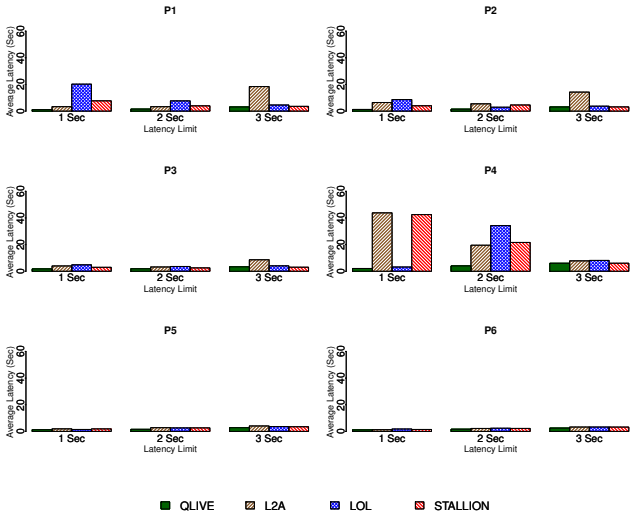


Figure 6: Average latency for algorithms under different latency limits and network profiles.

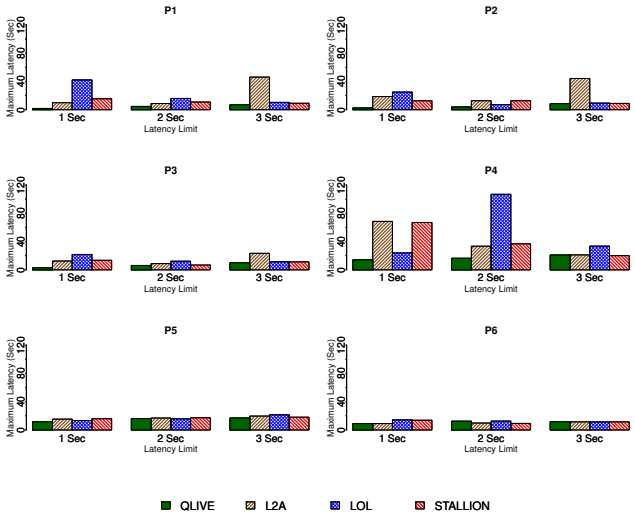


Figure 7: Maximum latency at any instance for algorithms under different latency limits and network profiles.

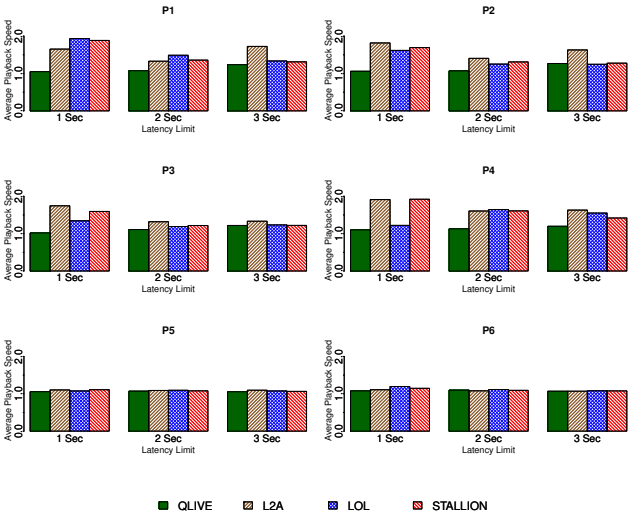


Figure 8: Average playback speed for algorithms under different latency limits and network profiles.

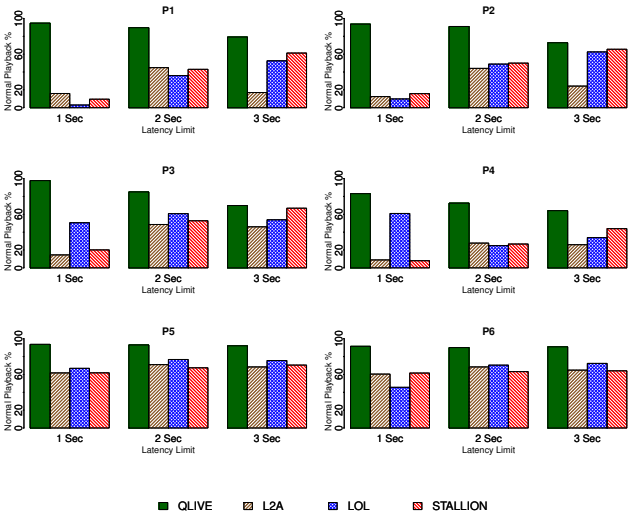


Figure 9: Normal Playback speed percentage for algorithms under different latency limits and network profiles.

for a small duration, causing the instantaneous maximum latency to be higher than the limits.

5.5.5 Playback speed. QLive and the compared algorithms modify the playback speed to control the latency within the given limits. Having lower latency is essential for QoE in live streaming, but a frequent change in playback speed hampers the QoE [33]. We analyze the average playback speed and the percentage of the total time a video is played at normal speed using the different algorithms in Figures 8 and 9 respectively. As we can see, the average playback speed for QLive is closer to the normal speed, i.e., between 1.01× to 1.26× for different latency limits and profiles. The percentage of time that QLive plays at normal speed is also the highest, at 64% - 97%. Interestingly, this percentage declines with the increase in the latency limit. This decline is because we use 1-, 2-, and 3-second segments for a 3-second latency limit. Profiles P3 to P6 have a bandwidth that changes every second. Having a 3-second segment makes the adaptation difficult, as the player changes the playback speed and representation level only at the segment boundary. Therefore, in the case of 1-second latency limit where there is just a 1-second segment to test, QLive average playback speed is closer to the normal speed and plays at normal speed for the longest duration.

The playback speed adaptation for other compared algorithms causes them to play longer at a speed other than 1× because it considers the gap between the latency limits and the instantaneous value of latency, regardless of bandwidth and the buffer condition. Therefore, when the instantaneous latency is above the limit, and the buffer is draining, an increase in playback speed causes rebuffering and adds to the latency. L2A, in general, seldom plays back the video at 1× speed, even for the 1-second segment. The duration of playback at normal speed for L2A is less than 20% of the total playback time in many cases.

5.5.6 Quality of experience. We measured QoE using two state-of-art QoE models. The first model is proposed by Yin et al. [62] as follows:

$$QoE = \sum_{n=1}^{\mathcal{N}} q(R_n) - \lambda \sum_{n=1}^{\mathcal{N}-1} |q(R_{n+1}) - q(R_n)| - \Delta T_{stall} - \Delta_s T_s. \quad (4)$$

Here, $\mathcal{N}$ is the total number of segments played for each experimental cycle; q maps a bitrate to a quality value; T_{stall} is the total stall duration during the playback, and T_s is the startup delay. As in [62], $\lambda = 1$, Δ and Δ_s are set to the maximum bitrate of the video sample. $q(\cdot)$ is the identity function for bitrate R_n. These parameters ensure that any increase in bitrate at the cost of an increase in changes of representations or increase of stalls results in negative QoE.

The second QoE model [3] extends the first model to capture the effect of playback speed and latency. It is used in Twitch's ACM MMSys 2020 Grand Challenge. The model is

$$\begin{aligned} QoE = &\sum_{n=1}^{\mathcal{N}} (q(R_n) - \phi L_n - \sigma|1 - P_n| - \Delta T_n - \Delta_s T_s) \\ &- \lambda \sum_{n=1}^{\mathcal{N}-1} |q(R_{n+1}) - q(R_n)|. \end{aligned} \quad (5)$$

The model additionally penalizes the QoE for latency L, and for non-normal speed $|1 - P_n|$ for each segment n (P_n is the playback speed of segment n). Similar to the first model, we set ϕ and σ as the maximum bitrate so that any increase in bitrate at the cost of latency or change in the playback speed results in the negative QoE.

Figures 10 and 11 shows this QoE for different algorithms using Equation (4), and (5) respectively. QoE is highest with Profiles P5 and P6, and lowest with profile P3 and P4 for all four algorithms. This is not surprising since Profiles P5 and P6 has the highest bandwidth, while P3 and P4 has the lowest.

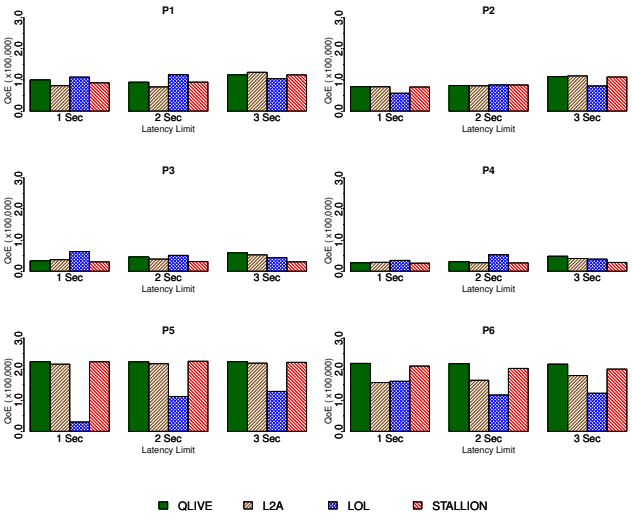


Figure 10: QoE for algorithms under different latency limits and network profiles based on bitrate, changes in representations, and stalls.

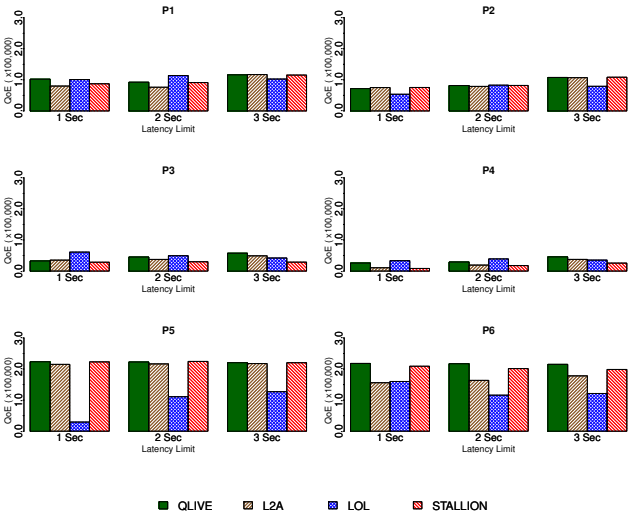


Figure 11: QoE for algorithms under different latency limits and network profiles based on bitrate, changes in representations, stalls, playback speed and latency.

QLive performs consistently well in both QoE metrics. It is either the highest or is very close to the highest QoE value is almost all of the profiles and latency limit setting. It has up to 47% better QoE than the compared methods using both QoE equations. Other algorithms have highly varying QoE as their performance also a high variation for different performance metrics across different profiles and latency limits.

5.5.7 Results over Internet. We tested QLive with the 1-second latency limit over the Internet, where the encoding server and the client are on two different continents and approximately 10900 km away. We ran ten sessions, each of 1-hour duration, during the COVID-19 pandemic period when the Internet traffic is experiencing an unexpected surge as most of the events and meetings are happening online. As a result, many streaming services in the UK and Europe are forced to reduce the video quality to prevent a possible collapse of the Internet [49]. In this overloaded Internet situation, QLive achieved an average latency of 0.92 seconds with an average stall of 1.6 seconds while playing at the unit speed for 99.2% of the total time. The average bitrate measured is 3837 Kbps, where the maximum attainable value is 4000 Kbps as per the encoding. Unlike the playback speed control used in `Dash.js` and other compared algorithms, QLive does not slow down the playback speed to reach the latency limit. Our playback speed adaptation changes the playback speed when the instantaneous latency is more than the limit or buffer occupancy is too low to avoid stalls. Therefore, QLive plays the video at a latency lower than the limit for a prolonged period when the bandwidth is sufficient.

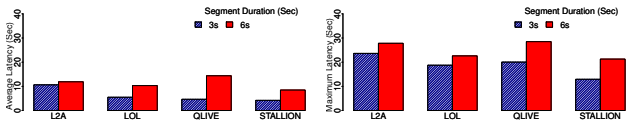


Figure 12: Average (left) and maximum (right) latency using 3- and 6-second segments and 3-second latency limit.

5.5.8 Segment duration versus latency limit. To analyze the effect of using a segment longer than the latency limit, we performed an experiment using 6-second segments with 3-second latency limit and compared the maximum instantaneous latency and average latency for all the algorithms. Figure 12 compares the average and maximum instantaneous latency using 3- and 6-seconds segments. Since the changes in representation are only possible at the segment boundaries, having a longer segment may result in more erroneous decisions for rate and playback speed adaptation. Further, having a segment longer than the latency limit leads to an increase in the segment's playback speed whose encoding is still in progress, causing a buffer drought and further increases the latency.

The average latency for QLive jumps more than 3 times when the segment duration is 6 seconds compared to 3 seconds. L2A is minimally affected as its original latency of 3 seconds was already the highest compared to other algorithms. LoL and Stallion also face 5 seconds and 4 seconds increase in average latency. Similarly, the maximum latency also increased for all algorithms by 4 to 9 seconds when we increase the segment duration.

5.5.9 Summary. QLive performs better in most performance metrics compared to the state-of-the-art methods because of its combined rate and playback speed adaptation algorithm. The default `Dash.js` playback speed adaptation algorithm used by the compared bitrate adaptation algorithm is not suitable when the buffer is low, and the latency is beyond the limit. Increasing the playback speed in such a situation further increases the latency and causes more stalls. Despite having more changes in representation levels, QLive has the highest bitrate with minimal stall, while playing the video longer at normal speed. Other ABR algorithms (LoL, L2A,

and Stallion) have better performance in terms of the number of changes in the representations. However, the network situation in profile P3 to P6, where the bandwidth fluctuations are high, creates a challenging situation that needs more sensitive bitrate and playback adaptation.

6 CONCLUSION

We showed that a DASH-based low latency live video streaming system could control the latency for a given target without skipping the frames and with an occasional increase in playback speed while playing at the normal speed most of the time. The key techniques that enable this are (*i*) an accurate application-level bandwidth estimation technique, and (*ii*) an analytical queuing model that relates playback speed to other system parameters, including the buffering latency, estimated bandwidth, and representation bitrate. We also showed that despite the widespread belief about CMAF with CTE making the latency independent of the segment, the segment's duration still plays a significant role in the instantaneous value of latency. This characteristic helped us formulate the policy of keeping the segment duration less than or equal to the target latency limit for efficiently controlling the latency. The proposed QLive bitrate adaptation is generic and can be extended to a non-live scenario where a user may fast-forward a video or play the video back in slow motion. One limitation of our work is that there are no well-established QoE metrics that consider the effect of playback speed on video quality, stall, and latency according to the video content type and its application. Designing such a QoE metric needs detailed user study for specific use cases and remains an open problem for further research and development.

ACKNOWLEDGMENTS

This work is part of NExT++ research, supported by the National Research Foundation, Prime Minister's Office, Singapore under its IRC@SG Funding Initiative and is partially supported by the Singapore Ministry of Education Academic Research Fund Tier 1 (T1 251RES2038)

REFERENCES

[1] Ramon Aparicio-Pardo, Karine Pires, Alberto Blanc, and Gwendal Simon. 2015. Transcoding live adaptive video streams at a massive scale in the cloud. In *Proceedings of the 6th ACM Multimedia Systems Conference (MMSys '15)*. ACM, Portland, Oregon, 49–60.

[2] Thomas Barnett Jr., Shruti Jain, Usha Andra, and Khurana Taru. 2018. Cisco Visual Networking Index (VNI)Complete Forecast Update, 2017–2022. [Online]. Available: https://www.cisco.com/c/dam/m/en_us/network-intelligence/service-provider/digital-transformation/knowledge-network-webinars/pdfs/1211_BUSINESS_SERVICES_CKN_PDF.pdf.

[3] Abdelhak Bentaleb. 2020. QoE Evaluation for CMAF-based Low-latency Streaming. [Online]. Available: https://github.com/twitchtv/acm-mmsys-2020-grand-challenge/blob/master/NQoE.pdf.

[4] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann. 2019. A Survey on Bitrate Adaptation Schemes for Streaming Media over HTTP. *IEEE COMST* 21, 1 (2019), 562–585. https://doi.org/10.1109/COMST.2018.2862938

[5] Abdelhak Bentaleb, Christian Timmerer, Ali C Begen, and Roger Zimmermann. 2019. Bandwidth prediction in low-latency chunked streaming. In *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '19)*. ACM, Amherst, Massachusetts, 7–13.

[6] Abdelhak Bentaleb, Christian Timmerer, Ali C Begen, and Roger Zimmermann. 2020. Performance Analysis of ACTE: a Bandwidth Prediction Method for Low-Latency Chunked Streaming. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 16, 2s (2020), 1–24.

[7] Olga Bondarenko, Koen De Schepper, Ing-Jyh Tsang, Bob Briscoe, Andreas Petlund, and Carsten Griwodz. 2016. Ultra-low delay for all: Live experience, live analysis. In *Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16)*. ACM, Klagenfurt, Austria, Article 33, 4 pages.

[8] Nassima Bouzakaria, Cyril Concolato, and Jean Le Feuvre. 2014. Overhead and performance of low latency live streaming using MPEG-DASH. In *Proceedings of the 5th International Conference on Information, Intelligence, Systems and Applications (IISA '14)*. IEEE, Chania, Crete, Greece, 92–97.

[9] Olivier Brun and Jean-Marie Garcia. 2000. Analytical solution of finite capacity M/D/1 queues. *Journal of Applied Probability* 37, 4 (2000), 1092–1098.

[10] Joachim Bruneau-Queyreix, Mathias Lacaud, and Daniel Négru. 2017. A Hybrid P2P/Multi-Server Quality-Adaptive Live-Streaming Solution Enhancing End-User's QoE. In *Proceedings of the 25th ACM international conference on Multimedia (MM '17)*. ACM, Mountain View, California, USA, 1261–1262.

[11] Xu Cheng and Jiangchuan Liu. 2010. Tweeting videos: Coordinate live streaming and storage sharing. In *Proceedings of the 20th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '10)*. ACM, Amsterdam, The Netherlands, 15–20.

[12] Simon Da Silva, Joachim Bruneau-Queyreix, Mathias Lacaud, Daniel Négru, and Laurent Réveillère. 2018. MUSLIN demo: High QoE fair multi-source live streaming. In *Proceedings of the 9th ACM Multimedia Systems Conference (MMSys '18)*. ACM, Amsterdam, Netherlands, 529–532.

[13] Dacast. 2019. What are the recommended encoder settings for my stream? [Online]. Available: https://www.dacast.com/support/knowledgebase/what-are-the-recommended-encoder-settings-for-my-stream/.

[14] Luca De Cicco and Saverio Mascolo. 2010. An experimental investigation of the Akamai adaptive video streaming. In *Proceedings of the 6th International Conference on HCI in Work and Learning, Life and Leisure: Workgroup Human-Computer Interaction and Usability Engineering (USAB'10)*. Springer-Verlag, Klagenfurt, Austria, 447–464.

[15] Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano. 2011. Feedback control for adaptive live video streaming. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSys '11)*. ACM, San Jose, CA, USA, 145–156.

[16] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. 2011. Understanding the Impact of Video Quality on User Engagement. In *Proceedings of the 2011 ACM SIGCOMM Conference (SIGCOMM '11, Vol. 41)*. ACM, Toronto, Ontario, Canada, 362–373.

[17] Kristian Evensen, Tomas Kupka, Dominik Kaspar, Pål Halvorsen, and Carsten Griwodz. 2010. Quality-adaptive scheduling for live streaming over multiple access networks. In *Proceedings of the 20th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '10)*. ACM, Amsterdam, The Netherlands, 21–26.

[18] Facebook. 2019. What are the video format guidelines for live streaming on Facebook? [Online]. Available: https://www.facebook.com/help/1534561009906955.

[19] Tongtong Feng, Haifeng Sun, Qi Qi, Jingyu Wang, and Jianxin Liao. 2019. Vabis: Video Adaptation Bitrate System for Time-critical Live Streaming. *IEEE Transactions on Multimedia* 22, 11 (2019), 2963 – 2976.

[20] Roy Fielding and Julian Reschke. 2014. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. [Online]. Available: https://tools.ietf.org/html/rfc7230.

[21] Flowplayer. 2019. Internet connection and recommended encoding settings. [Online]. Available: https://support.video.ibm.com/hc/en-us/articles/207852117-Internet-connection-and-recommended-encoding-settings.

[22] Flowplayer. 2019. Recommended Livestream settings. [Online]. Available: https://flowplayer.com/help/livestreaming/recommended-live-stream-settings.

[23] International Organization for Standardization and CH International Electrotechnical Commission, Geneva. 2019. Information technology–Multimedia application format (MPEG-A) – Part19: Common media application format (CMAF) for segmented media. Standard ISO/IEC 23000-19:2018. . [Online]. Available: https://www.iso.org/standard/71975.html.

[24] DASH Industry Forum. 2014. Guidelines for implementation: DASH-AVC/264 test cases and vectors. [Online]. Available: http://dashif.org/guidelines/.

[25] DASH Industry Forum. 2017. dash.js. [Online]. Available: https://github.com/Dash-Industry-Forum/dash.js/wiki.

[26] Giovanni Gualdi, Rita Cucchiara, and Andrea Prati. 2006. Low-Latency Live Video Streaming over Low-Capacity Networks. In *Proceedings of the 8th IEEE International Symposium on Multimedia (ISM '06)*. IEEE, San Diego, CA, USA, 449–456.

[27] Craig Gutterman, Brayn Fridman, Trey Gilliland, Yusheng Hu, and Gil Zussman. 2020. Stallion: video adaptation algorithm for low-latency video streaming. In *Proceedings of the 11th ACM Multimedia Systems Conference*. ACM, Istanbul, Turkey, 327–332.

[28] Antti Heikkinen, Pekka Pääkkönen, Marko Viitanen, Jarno Vanne, Tommi Riikonen, and Kagan Bakanoglu. 2018. Fast and easy live video service setup using lightweight virtualization. In *Proceedings of the 9th ACM Multimedia Systems Conference (MMSys '18)*. ACM, Amsterdam, The Netherlands, 487–489.

[29] Devin Heitmueller and Colleen. 2019. Streamline Low Latency DASH preview. [Online]. Available: https://github.com/streamlinevideo/low-latency-preview.

[30] Ruying Hong, Qiwei Shen, Lei Zhang, and Jing Wang. 2019. Continuous Bitrate & Latency Control with Deep Reinforcement Learning for Live Video Streaming. In *Proceedings of the 27th ACM International Conference on Multimedia (MM '19)*. ACM, Nice, France, 2637–2641.

[31] Rafael Huysegems, Jeroen Van Der Hooft, Tom Bostoen, Patrice Rondao Alface, Stefano Petrangeli, Tim Wauters, and Filip De Turck. 2015. HTTP/2-based methods to improve the live experience of adaptive streaming. In *Proceedings of the 23rd ACM International Conference on Multimedia (MM '15)*. ACM, Brisbane, Australia, 541–550.

[32] Theo Karagkioules, Rufael Mekuria, Dirk Griffioen, and Arjen Wagenaar. 2020. Online learning for low-latency adaptive streaming. In *Proceedings of the 11th ACM Multimedia Systems Conference*. ACM, Istanbul, Turkey, 315–320.

[33] Kazutaka Kurihara. 2012. CinemaGazer: A system for watching videos at very high speed. In *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI '12)*. ACM, Capri Island, Italy, 108–115.

[34] Will Law. 2018. Ultra-Low-Latency Streaming Using Chunked-Encoded and Chunked-Transferred CMAF. [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/white-paper/low-latency-streaming-cmaf-whitepaper.pdf.

[35] Will Law. 2018. Ultra-Low-Latency Streaming Using Chunked-Encoded and Chunked-Transferred CMAF. Akamai White paper. [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/white-paper/low-latency-streaming-cmaf-whitepaper.pdf.

[36] Jean Le Feuvre, Cyril Concolato, Nassima Bouzakaria, and Viet-Thanh-Trung Nguyen. 2015. MPEG-DASH for low latency and hybrid streaming services. In *Proceedings of the 23rd ACM International Conference on Multimedia (MM '15)*. ACM, Brisbane, Australia, 751–752.

[37] Yi J Liang and Bernd Girod. 2002. Low-latency streaming of pre-encoded video using channel-adaptive bitstream assembly. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '02, Vol. 1)*. IEEE, Lausanne, Switzerland, Switzerland, 873–876.

[38] May Lim, Mehmet N Akcay, Abdelhak Bentaleb, Ali C Begen, and Roger Zimmermann. 2020. When they go high, we go low: low-latency live streaming in dash.js with LoL. In *Proceedings of the 11th ACM Multimedia Systems Conference*. ACM, Istanbul, Turkey, 321–326.

[39] Britta Meixner, Jan Willem Kleinrouweler, and Pablo Cesar. 2018. 4G/LTE channel quality reference signal trace data set. In *Proceedings of the 9th ACM Multimedia Systems Conference (MMSys '18)*. ACM, Amsterdam, Netherlands, 387–392.

[40] Shabnam Mirshokraie and Mohamed Hefeeda. 2010. Live peer-to-peer streaming with scalable video coding and networking coding. In *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems (MMSys '10)*. ACM, Phoenix, Arizona, USA, 123–132.

[41] Mario Montagud, Fernando Boronat, Bernardino Roig, and Almanzor Sapena. 2017. How to perform AMP? Cubic adjustments for improving the QoE. *Computer Communications* 103 (2017), 61–73.

[42] Meinard Müller. 2007. Dynamic Time Warping. In *Information Retrieval for Music and Motion*, Jan Fagerberg, David C. Mowery, and Richard R. Nelson (Eds.). Springer, Chapter 4, 69–84.

[43] Ihsan Mert Ozcelik and Cem Ersoy. 2020. Low-Latency Live Streaming over HTTP in Bandwidth-Limited Networks. *IEEE Communications Letters* (2020), 1–1.

[44] Huan Peng, Yuan Zhang, Yongbei Yang, and Jinyao Yan. 2019. A Hybrid Control Scheme for Adaptive Live Streaming. In *Proceedings of the 27th ACM International Conference on Multimedia (MM '19)*. ACM, Nice, France, 2627–2631.

[45] Stefano Petrangeli, Jeroen van der Hooft, Tim Wauters, Rafael Huysegems, Patrice Rondao Alface, Tom Bostoen, and Filip De Turck. 2016. Live streaming of 4K ultra-high definition video over the Internet. In *Proceedings of the 7th International Conference on Multimedia Systems*. ACM, Klagenfurt, Austria, 1–4.

[46] Pallavi Powale. 2017. YouTube Engineering and Developers Blog - Variable speed playback on mobile. [Online] https://youtube-eng.googleblog.com/2017/09/variable-speed-playback-on-mobile.html.

[47] Benjamin Rainer and Christian Timmerer. 2014. A quality of experience model for Adaptive Media Playout. In *Sixth International Workshop on Quality of Multimedia Experience (QoMEX '14)*. IEEE, Singapore, 177–182.

[48] Benjamin Rainer and Christian Timmerer. 2014. A subjective evaluation using crowdsourcing of Adaptive Media Playout utilizing audio-visual content features. In *IEEE Network Operations and Management Symposium (NOMS '14)*. IEEE, Krakow, Poland, 1–7.

[49] Ian Randall. 2020. YouTube and Amazon Prime Video follow Netflix in slashing the quality of their video streams across Europe to prevent the internet collapsing under the strain of unprecedented usage during the coronavirus pandemic. [Online]. Available: https://www.dailymail.co.uk/sciencetech/article-8133833/YouTube-reduce-streaming-quality-Europe-coronavirus.html.

[50] Devdeep Ray, Jack Kosaian, KV Rashmi, and Srinivasan Seshan. 2019. Vantage: Optimizing video upload for time-shifted viewing of social live streams. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. ACM, Beijing, China, 380–393.

[51] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. 2013. Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications. In *Proceedings of the 4th ACM Multimedia Systems Conference (MMSys '13)*. ACM, Oslo, Norway, 114–118.

[52] Roberto Roverso, Riccardo Reale, Sameh El-Ansary, and Seif Haridi. 2015. SmoothCache 2.0: CDN-quality adaptive HTTP live streaming on peer-to-peer overlays. In *Proceedings of the 6th ACM Multimedia Systems Conference (MMSys '15)*. ACM, Portland, Oregon, 61–72.

[53] Zhijie Shen and Roger Zimmermann. 2011. LAN-awareness: Improved P2P live streaming. In *Proceedings of the 21st International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '11)*. ACM, Vancouver, British Columbia, Canada, 3–8.

[54] E. Steinbach, N. Farber, and B. Girod. 2001. Adaptive playout for low latency video streaming. In *Proceedings of the 2001 International Conference on Image Processing (ICIP '01, Vol. 1)*. IEEE, Thessaloniki, Greece, 962–965.

[55] Viswanathan Swaminathan and Sheng Wei. 2011. Low latency live video streaming using HTTP chunked encoding. In *Proceedings of the 13th IEEE International Workshop on Multimedia Signal Processing (MMSP '11)*. IEEE, Hangzhou, China, 1–6.

[56] Raymond Sweha, Vatche Ishakian, and Azer Bestavros. 2012. Angelcast: Cloud-based peer-assisted live streaming using optimized multi-tree construction. In *Proceedings of the 3rd Multimedia Systems Conference (MMSys '12)*. ACM, Chapel Hill, North Carolina, 191–202.

[57] Jeroen van der Hooft, Dries Pauwels, Cedric De Boom, Stefano Petrangeli, Tim Wauters, and Filip De Turck. 2018. Low-Latency Delivery of News-Based Video Content. In *Proceedings of the 9th ACM Multimedia Systems Conference (MMSys '18)*. ACM, Amsterdam, The Netherlands, 537–540.

[58] David Varodayan and Wai-tian Tan. 2011. Error-resilient live video multicast using low-rate visual quality feedback. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSys '11)*. ACM, San Jose, CA, USA, 233–244.

[59] Miao Wang, Lisong Xu, and Byrav Ramamurthy. 2009. Providing statistically guaranteed streaming quality for peer-to-peer live streaming. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '09)*. ACM, Williamsburg, VA, USA, 127–132.

[60] Praveen Kumar Yadav, Arash Shafiei, and Wei Tsang Ooi. 2017. QUETRA: A queuing theory approach to DASH rate adaptation. In *Proceedings of the 25th ACM International Conference on Multimedia (MM '17)*. ACM, Mountain View, California, USA, 1130–1138.

[61] Mariem Ben Yahia, Yannick Le Louedec, Gwendal Simon, Loutfi Nuaymi, and Xavier Corbillon. 2019. HTTP/2-based frame discarding for low-latency adaptive video streaming. *ACM Transactions on Multimedia Computing, Communications, and Applications* 15, 1 (2019), 1–23.

[62] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. ACM, London, United Kingdom, 325–338.

[63] YouTube. 2019. Choose live encoder settings, bitrates, and resolutions. [Online]. Available: https://support.google.com/youtube/answer/2853702?hl=en.

[64] Guanghui Zhang and Jack YB Lee. 2019. LAPAS: Latency-aware Playback-Adaptive Streaming. In *Proceedings of the 2019 IEEE Wireless Communications and Networking Conference (WCNC '19)*. IEEE, Marrakesh, Morocco, 1–6.

[65] Rui-Xiao Zhang, Tianchi Huang, Ming Ma, Haitian Pang, Xin Yao, Chenglei Wu, and Lifeng Sun. 2019. Enhancing the crowdsourced live streaming: A deep reinforcement learning approach. In *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '19)*. ACM, Amherst, Massachusetts, 55–60.

[66] Yin Zhao, Qi-Wei Shen, Wei Li, Tong Xu, Wei-Hua Niu, and Si-Ran Xu. 2019. Latency Aware Adaptive Video Streaming using Ensemble Deep Reinforcement Learning. In *Proceedings of the 27th ACM International Conference on Multimedia (MM '19)*. ACM, Nice, France, 2647–2651.

[67] Fen Zhou, Shakeel Ahmad, Eliya Buyukkaya, Raouf Hamzaoui, and Gwendal Simon. 2012. Minimizing server throughput for low-delay live streaming in content delivery networks. In *Proceedings of the 22nd International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '12)*. ACM, Toronto, Ontario, Canada, 65–70.