Bandwidth Prediction in Low-Latency Chunked Streaming

Abdelhak Bentaleb[★⋆], Christian Timmerer[+], Ali C. Begen[*], Roger Zimmermann[★]
[★]National University of Singapore, [+]Alpen-Adria Universität & Bitmovin Inc., [*]Ozyegin University & Networked Media
{bentaleb,rogerz}@comp.nus.edu.sg, christian.timmerer@itec.uni-klu.ac.at, ali.begen@ozyegin.edu.tr

ABSTRACT

HTTP adaptive streaming with chunked transfer encoding can be used to offer low-latency streaming without sacrificing the coding efficiency. While this allows a media segment to be generated and delivered at the same time, which is critical in reducing the latency, the conventional bitrate adaptation schemes make often grossly inaccurate bandwidth measurements due to the presence of idle periods between the chunks. These wrong measurements cause the streaming client to make bad adaptation decisions. To this end, we design ACTE, a new bitrate adaptation scheme that leverages the unique nature of chunk downloads. ACTE uses a sliding window to accurately measure the available bandwidth and an online linear adaptive filter to predict the bandwidth into the future. Results show that ACTE achieves 96% measurement accuracy, which translates to a 65% reduction in the number of stalls and a 49% increase in quality of experience on average compared to other schemes.

CCS CONCEPTS

• **Information systems → Multimedia streaming**.

KEYWORDS

HAS; ABR; DASH; CMAF; low-latency; HTTP chunked transfer encoding; bandwidth measurement and prediction; RLS.

ACM Reference Format:
Abdelhak Bentaleb, Christian Timmerer, Ali C. Begen, Roger Zimmermann. 2019. Bandwidth Prediction in Low-Latency Chunked Streaming. In *29th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'19), June 21, 2019, Amherst, MA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3304112.3325611

1 INTRODUCTION

HTTP adaptive streaming (HAS) has become the common technology to offer viewers a great quality of experience (QoE) [4]. In HAS, the media is divided into short-duration segments that are encoded at different bitrates and resolutions (*i.e.*, representations) and offered as HTTP objects to the streaming clients. On the client side, the adaptive bitrate (ABR) logic picks segments from a suitable representation and fetches them from the server.

[*]A. Bentaleb did this work when he was an intern at Bitmovin Inc.

NOSSDAV'19, June 21, 2019, Amherst, MA, USA

ACM ISBN 978-1-4503-6298-6/19/06…$15.00
https://doi.org/10.1145/3304112.3325611

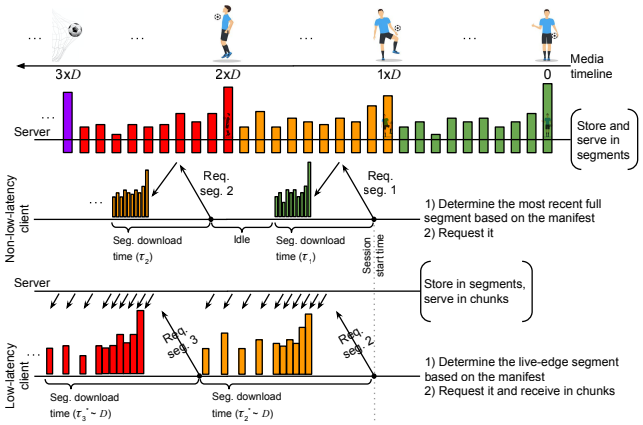


Figure 1: The S_i/τ_i formula works well for the non-low-latency client but not for the low-latency client. D indicates the segment duration. Segments are colored differently and each bar indicates a chunk with its size being proportional to the bar's height. Not to scale, for illustration purposes only.

Linear over-the-top (OTT) services using HAS are widely available today. Most services deliver an excellent experience when clients buffer several tens of seconds of media, but face issues when running in low-latency mode, where the delta time from the capture to rendering is expected to be five seconds or less. In HAS, capturing, encoding, packaging, publishing to the origin, delivering through a content delivery network (CDN), buffering, decoding and rendering incur serial delays. To keep the total latency under five seconds, all these processes need to be streamlined. One way to do this without having to use short segments, which would significantly reduce the coding efficiency, is to use HTTP chunked transfer encoding, where long segments can be generated and delivered concurrently in small pieces that are referred to as chunks [7, 10]. For example, a video segment can be six seconds long, but chunking could be done even at the frame level, which drastically reduces the serialization delay, and hence, the latency in linear delivery. HTTP chunked encoding is a standardized feature of HTTP/1.1 (RFC 7230). If the media is packaged according to the MPEG Common Media Application Format (CMAF; ISO/IEC 23000-19) [1] standard, both MPEG's Dynamic Adaptive Streaming over HTTP (DASH) [14] and Apple's HTTP Live Streaming (HLS) [11] can use this feature [3].

ABR schemes that were designed for non-low-latency streaming download the segments that have been already packaged and are available on the server. In this case, the well-known S_i/τ_i formula, where S_i is the size of the segment i and τ_i is the segment download time (the difference between when segment i was fully received and the request was sent), provides a good approximation for the available bandwidth. On the other hand, a low-latency ABR scheme must download the *live-edge segment* that is still being prepared (encoded and packaged) at the time of the request. In this case, the

chunks of the segment that are already available are sent to the client as fast as possible (transmission is *network-limited*), whereas the portions that are yet to be prepared will be sent to the client as they become available (transmission will be *source-limited*), leaving idle periods between the chunks, as illustrated in Figure 1. The presence of idle periods causes the S_i/τ_i formula to fail [10].

In the low-latency mode (bottom of Figure 1), chunks in a segment are delivered with relatively more consistent timing ($\tau_i^* \sim D$) since the transmission is source-limited. Note that if the network cannot sustain the selected bitrate, the transmission will be network-limited and the ABR logic naturally needs to downshift to a lower bitrate. That is, the download time for a segment cannot be shorter than its duration (except for the first few segments, which could be partially available already when the client's request has been received by the server). Thus, an ABR scheme naively using the S_i/τ_i^* formula will always find a value equal to (or slightly smaller than) the segment encoding bitrate. This prevents the client from switching to a higher bitrate representation even when there is enough available bandwidth in the network to support higher bitrate representations (cf. Section 3.1 for more details). Clearly, we need a new chunk-aware bandwidth measurement method that takes the idle periods between the chunks into account.

This paper presents ACTE: **A**BR for **C**hunked **T**ransfer **E**ncoding. ACTE performs accurate bandwidth measurement and prediction in low-latency streaming, and relies on three main components:

(1) *Bandwidth Measurement* implements a sliding window based moving average method to measure chunk-level bandwidth.
(2) *Bandwidth Prediction* implements an online linear adaptive filter based on an adaptive recursive least squares (RLS) algorithm [8] to predict the bandwidth into the future based on the measured samples. RLS works well since bandwidth measurements are correlated, stationary and deterministic.
(3) *ABR Controller* implements a throughput-based bitrate selection logic that takes bandwidth prediction values as input and computes the best bitrate to select.

We implemented ACTE in the dash.js reference player [6] and offer a demo [2]. We analyzed ACTE's performance on a broad set of real-world network traces with different video parameters, bandwidth measurement algorithms and dash.js ABR schemes.

The rest of the paper is organized as follows. Related work is provided in Section 2, followed by the details for the ACTE scheme and its implementation in Section 3. Section 4 presents the experimental evaluation and Section 5 concludes the paper.

2 RELATED WORK

Swaminathan *et al.* [16] proposed a low-latency HTTP chunked encoding based approach that decoupled the live delay from the segment duration. The approach analytically evaluated the latency in three live streaming methods, including a (*i*) segment-based method where the player requested the segment only if it was fully available, (*ii*) server-wait method where the player requested a segment before it was ready at the server, and (*iii*) chunked encoding method where the player received chunks before the full segment was ready. Bouzakaria *et al.* [5] and El Essaili *et al.* [7] investigated the adoption of chunked transfer encoding in live DASH delivery systems to reduce latency. The former introduced a new timing parameter in the DASH manifest that assisted the players in their requests, while the latter developed an HTML5 MSE player with CMAF low-latency support. Houzé *et al.* [9] designed a low-latency solution that benefits from multipath capabilities and chunked transfer encoding. It splits and downloads multiple video frames from different paths taking the frame sizes into account. Shuai *et al.* [13] analyzed the impact of the player buffer occupancy on live streaming and proposed an ABR scheme based on buffer stabilization for low latency with two-second segments. van der Hooft *et al.* [17] described an HTTP/2 push-based approach for low-latency live streaming with super short segments without using CMAF or chunked transfer encoding. Regarding the implementation of CMAF low latency, DASH-IF and Akamai provided a proof-of-concept in the dash.js player [10]. For more details about different ABR schemes, refer to [4].

The studies mentioned above show good performance in certain network environments and under certain assumptions. However, none of them has systematically analyzed how to effectively select a good bitrate in the context of various network conditions with the explicit aim of maintaining a low latency while utilizing CMAF.

3 THE ACTE SCHEME

3.1 Low-Latency Data Collection

To understand the effects of ABR and its behavior with low-latency streaming, we collected data of a real-time video session from the Twitch platform with low-latency mode enabled. We used Wireshark and our Web crawler based on the Twitch API-v5[1] to collect packet and application-level data, respectively. There were two reasons for selecting Twitch: (*i*) its CMAF low-latency support, and (*ii*) the Twitch API-v5, which allows to collect various live streaming session information and metrics like bitrate, resolution, buffer size, total latency, *etc.*, in real time.

We used a Scrapy-based[2] crawler to collect twelve hours of Twitch data from a channel at different times during a week in November 2018. We logged the data in a CSV file every 100 milliseconds. During the data collection, we ran traffic shaping on a proxy using tc-NetEm[3] to emulate a real last-mile link with varying network conditions (profile BW_3, see Table 2). The proxy was located on the client side and played the role of an intermediary between the client and Twitch server. To measure the bandwidth (= S_i/τ_i^*), we used the packet-level data collected with Wireshark, in particular, the segment sizes as well as segment request and arrival times. The video of the live channel was encoded at six bitrate levels of {0.18,0.73,1.83,2.5,3.1,8.8} Mbps with three resolutions of {540p,720p,1080p} conforming to CMAF.

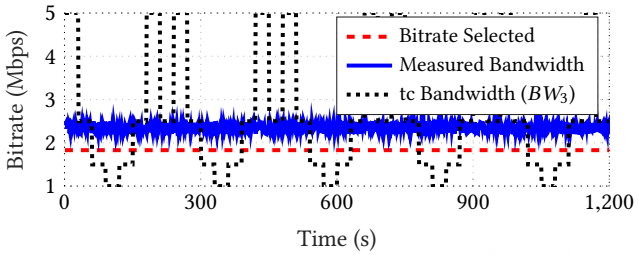


Figure 2: Results for CMAF low latency (Twitch).

[1] https://dev.twitch.tv/
[2] https://scrapy.org/
[3] https://wiki.archlinux.org/index.php/advanced_traffic_control

Figure 2 depicts twelve minutes of the collected data in terms of the selected bitrate, measured bandwidth (segment-based), and tc bandwidth. As the figure shows, the Twitch player (version 2.6.31-39e00d61) always selected the same bitrate of 1.83 Mbps, while the measured bandwidth ranged from 2 to 2.5 Mbps. The measured bandwidth clearly does not track the tc bandwidth accurately, resulting in a constant bitrate selection (*i.e.*, no adaptation).

3.2 ACTE Components and Functions

Motivated by our observations in Section 3.1, we designed ACTE. Figure 3 shows the components of ACTE in the dash.js reference player [6]. It has four components, *(1)* bandwidth measurement, *(2)* bandwidth prediction, *(3)* ABR controller and *(4)* logger. The list of notations used in ACTE is given in Table 1.

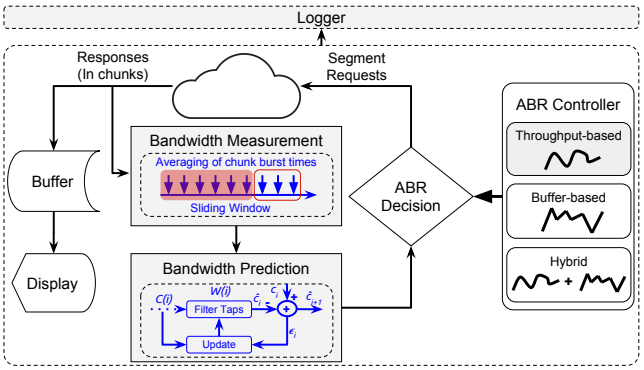


Figure 3: ACTE overview in dash.js reference player. Modifications required by ACTE are highlighted in gray.

Table 1: List of notations.

Notation	Meaning
$\mathcal{M}$	Filter length
$\mathcal{W}(i)$	Filter taps (coefficients) vector of length $\mathcal{M}$
$\mathcal{P}(i)$	The inverse correlation matrix of size $\mathcal{M} \times \mathcal{M}$
$\mathcal{G}(i)$	The gain vector of length $\mathcal{M}$
$C(i)$	Most recent bandwidth measurements of length $\mathcal{M}$
$\mathcal{R}$	List of bitrate levels
σ	Initial input variance estimate parameter for $\mathcal{P}(0)$
λ	Forgetting factor
i	Chunk downloading step
c_i	Measured average bandwidth at step i
ϵ_i	Estimated error at step i
$y_i(=\hat{c}_i)$	The bandwidth prediction at step i
$\hat{c}_{i+1}$	Bandwidth prediction for the next step (step $i+1$)
r_i	Bitrate level selected at step i
l_i	Current latency at step i
l_{target}	Target latency
B_i	Buffer level at step i
z	Sliding window size for bandwidth measurements
K	Total number of downloaded chunks
Q_m^n	Size of the m^{th} segment's n^{th} chunk
b_m^n	Beginning time of the chunk download
e_m^n	End time of the chunk download

3.2.1 Bandwidth Measurement. This component implements the sliding window moving average bandwidth measurement method, which computes the average bandwidth for the last z successful chunk downloads, where $z = 3$ in ACTE. At each chunk downloading step i, the average bandwidth is calculated as follows:

$$c_i = \frac{1}{z} \sum_{n=0}^{z-1} \frac{Q_m^{i-n}}{e_m^{i-n} - b_m^{i-n}}, \tag{1}$$

where Q is the chunk size, and b and e are the beginning and end times of the chunk download, respectively, as illustrated in Figure 4. (1) requires us to know the values for Q, b and e. Q is inferred from the HTTP header and the HTTP `Fetch` API provides us the value for e. However, with the standard HTTP protocol we have no means to determine the value for b. If there is a non-negligible idle period after a chunk is downloaded (*i.e.*, when $b_m^{n+1} - e_m^n \gg 0$), that chunk must be disregarded in computing (1). Since we do not know the b values, we have to determine such chunks in a different way.

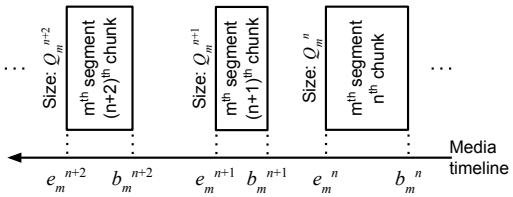


Figure 4: The idle periods between the chunks of a segment.

First, we assume that the server pushes the chunks at full network speed, and thus, the idle periods could only happen between the chunks of a particular segment, not within a chunk. We download every chunk into a partial buffer such that the sum of all partial buffers constitutes the respective segment. For each chunk n (for segment m), we divide the size of the partial buffer (Q_m^n) by the delta time between the previous chunk's end time (e_m^{n-1}) and the current chunk's end time (e_m^n). If this download rate is close to the average download rate of the segment (S_m/τ_m^*), it implies there is non-negligible idle period from e_m^{n-1} to e_m^n, and this chunk must be disregarded in computing (1). If not, the idle period is negligible and the chunk download rate is a good approximation of the available bandwidth, thus, should be used in (1).

3.2.2 Bandwidth Prediction. This component implements a linear history-based prediction, in particular, an online linear adaptive filter that is based on a recursive least squares (RLS) approach [8]. This procedure is presented in Algorithm 1, where for each chunk downloading step i, it takes as an input a vector of the $\mathcal{M}$ most recent bandwidth measurement values $C(i)$ given by the bandwidth measurement component (BW-Measurement), and then recursively computes the gain vector $\mathcal{G}(i)$, the inverse correlation matrix $\mathcal{P}(i)$ of the bandwidth measurement values, and the estimated error ϵ_i. These values are then used to update the filter taps vector $\mathcal{W}(i)$ of length $\mathcal{M}$, and finally return the future bandwidth prediction for the next step $\hat{c}_{i+1}$ (= $\mathcal{W}(i)$ $C(i)$) (see line 12 in Algorithm1). We used RLS for three main reasons. First, it is frequently used in practical real-time systems when there is a limitation in computational capabilities. Second, it does not need a prediction model for its input, and third, it leverages the fact that measured bandwidth values are correlated, stationary and deterministic. Moreover, the proposed algorithm is robust against time-varying network conditions through the forgetting factor λ, where this factor does not rely on the time-evolution model for the system, and in contrast, gives exponentially less weight to the past bandwidth measurements. For accurate bandwidth prediction, our algorithm strives to minimize the error ($\min \epsilon_i$) between the

next (step $i+1$) bandwidth measurement and the current (step i) future bandwidth prediction. The computational complexity and overhead of our algorithm is low depending on the filter length $\mathcal{M}$. The overall complexity is $O(\mathcal{M}^2)$ per time iteration.

Algorithm 1 Bandwidth Prediction (RLS-based)

$\mathcal{W}(0) = 0$, $\mathcal{P}(0) = \mathrm{I} \times \sigma^{-1}$, $\epsilon = 0$, $C(0) = 0$
$\sigma = 0.001$, $\lambda = 0.999$, BW-Measurement = `Sliding-Window()`
function ACTEUpdate()
  for Each chunk downloading step $i > 0$ **do**
    $\mathcal{G}(i) = \frac{\lambda^{-1}\mathcal{P}(i-1)C(i-1)}{1+C^T(i-1)\mathcal{P}(i-1)C(i-1)}$
    $\mathcal{P}(i) = \lambda^{-1}\mathcal{P}(i-1) - \lambda^{-1}\mathcal{G}(i)\, C^T(i-1)\, \mathcal{P}(i-1)$
    c_i = `Sliding-Window()`
    $y_i = \hat{c}_i = \mathcal{W}^T(i-1)\, C(i-1)$
    $\epsilon_i = c_i - y_i$
    $C(i)$ = `Update`(c_i)
    $\mathcal{W}(i) = \mathcal{W}(i-1) + \epsilon_i \mathcal{G}(i)$
    $\hat{c}_{i+1} = \mathcal{W}(i)\, C(i)$
    `Return`$(\hat{c}_{i+1})$

3.2.3 ABR Controller. This component implements three main heuristic-based ABR schemes that are (1) throughput-based: using the measured bandwidth to perform the ABR decisions, (2) buffer-based (BOLA): using the playback buffer occupancy [15] to perform the ABR decisions, and (3) hybrid: a combination of throughput-based and buffer-based. It also monitors the playback buffer occupancy through the buffer controller component to avoid stalls and maintains the buffer occupancy within a safe region (between minimum and maximum predefined thresholds for CMAF low latency). ACTE relies on the throughput-based ABR scheme, which is in turn based on the bandwidth measurement and bandwidth prediction components described above. At every chunk downloading step i, the ACTE ABR scheme takes the bandwidth prediction value $\hat{c}_{i+1}$ as input and then outputs the optimal bitrate level. The objective function $\mathcal{F}$ is defined as:

$$\mathcal{F}: \begin{cases} \textbf{find } r_i^{\star}, \arg\min \epsilon_i \text{ and } \arg\max \mathrm{QoE}_i \quad \forall i > 0, \\ \text{s.t.} \quad l_i \leq l_{target}, \\ \qquad c_i \approx y_i, \text{ and } r_i^{\star} \approx \hat{c}_{i+1} \\ \qquad 0.5 \leq B_i \leq l_{target}, \end{cases} \tag{2}$$

where the objective is to find the optimal bitrate level $r_i^{\star} \in \mathcal{R}$ that minimizes the estimated error and maximizes the QoE respecting the target latency $l_{target} \in [2.5, \ldots, 3.2]$ seconds (keeping the current latency l_i sufficiently small), current network capacity and current playback buffer occupancy B_i. We note that ACTE performs bitrate changes at segment boundaries, not chunk boundaries.

3.2.4 Logger. This component periodically records various metrics such as the selected bitrate, buffer occupancy, measured/predicted bandwidth values, stalls and startup delay.

4 EXPERIMENTAL EVALUATION

We evaluated the performance of ACTE through a set of real-world trace-driven experiments. Each experiment covers various operating regimes and settings including (*i*) video parameters: bitrate level, resolution, chunk and segment duration, and target latency, (*ii*) bandwidth variation profiles (see Table 2), (*iii*) bandwidth measurement and prediction parameters, (*iv*) bandwidth measurement algorithms: segment-based last bandwidth (SLBW), chunk-based exponentially weighted moving average (EWMA), chunk-based sliding window moving average (SWMA) and Will's simple slide-load (WSSL) [10], (*v*) ABR schemes of throughput-based (TH), buffer-based (BOLA) and hybrid (TH+BOLA, referred to as Dynamic), and (*vi*) QoE metrics of average bitrate, startup delay, stall duration and number of bitrate switches.

4.1 Methodology

4.1.1 Bandwidth Variation Profiles. As shown in Table 2, we used three network profiles in the evaluation, denoted BW_1, BW_2 and BW_3. These profiles correspond to the DASH-IF guidelines for a real-world bandwidth measurement dataset [6, 15] and follow the cascade pattern (high-low-high and low-high-low). Each profile consists of different bandwidth values, round-trip times (RTT; in milliseconds), packet loss rates (in %) and bandwidth variation durations (in seconds).

Table 2: Bandwidth variation profiles.

Network Profile	Bandwidth Values (Mbps)	Inter-variation Duration (s)
BW_1	4, 3.5, 3, 2.5, 3, 3.5	30
BW_2	5, 3, 2.5, 2, 2.5, 3, 5, 3	30
BW_3	5, 2.5, 1.5, 1, 1.5, 2.5, 5, 2.5	30

4.1.2 Video Parameters. We used the short movie *Tears of Steel*[4] for the evaluation. Each segment is six seconds long and encoded into three bitrate levels for video at {0.7, 1.3 and 2.0} Mbps and one bitrate level for audio at 96 Kbps. The chunk duration is 500 milliseconds, the target latency is set to 3.2 seconds, and each live streaming session is 560 seconds long. These video parameters are predefined based on an existing live service used for this experimental evaluation. In the future, we plan more extensive evaluations requiring a larger scale setup.

4.1.3 Bandwidth Measurement and Prediction Parameters. We ran an offline MATLAB-based experiment to find suitable parameter values (z, $\mathcal{M}$, σ, and λ) for the bandwidth measurement and prediction components that respected our objective function $\mathcal{F}$ and minimized the estimated error ϵ. We investigated the following values: z = [2 to 5], $\mathcal{M}$ = [2 to 5], σ = [0.001, 0.01, 0.1], and λ = [0.999, 0.5, 0.1]. Figure 5 shows the achieved estimated error as a function of time for ACTE (RLS-based) and Linear Mean Square (LMS) [8] with μ (mean) = 0.075. Using $z = 3$, $\mathcal{M} = 3$, $\sigma = 0.001$, and $\lambda = 0.999$, ACTE was quickly able to minimize the estimated error for bandwidth prediction, achieve the best result, and significantly outperform the LMS-based bandwidth prediction.

4.1.4 Performance Metrics. We tested the efficiency of our solution using four main models: *(1)* live latency [6], *(2)* Yin QoE [18], *(3)* ITU-T Rec. P.1203 QoE (Mode 0) [12], and *(4)* bandwidth prediction accuracy. The live latency model is given by the dash.js reference player [6] and represents the total time from capturing to rendering. The ITU-T Rec. P.1203 QoE model in Mode 0 takes as input four metrics: bitrate, stall duration, frame rate, and resolution. How to compute the QoE is described in [12]. This model outputs QoE values in the range of 1 to 5 (MOS), and we normalized

[4] https://mango.blender.org/download/

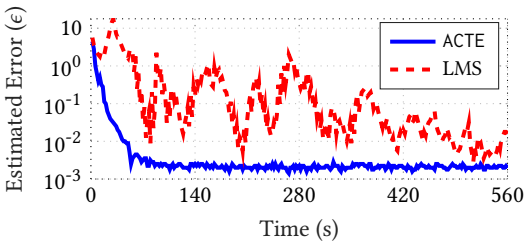


Figure 5: Estimated error in ACTE and LMS.

them (N-QoE^{itu}) to [0,1]. The Yin QoE model [18] combines four performance metrics into one QoE value as follows:

$$\sum_{i=1}^{K} q(r_i) - \delta_1 \sum_{i=1}^{K-1} \left| q(r_{i+1}) - q(r_i) \right| - \delta_2 T_{stall} - \delta_3 T_{sd}. \quad (3)$$

The four terms of (3) are the average bitrate level, number of bitrate switches, stall duration and startup delay, respectively. K ($= 560/0.5 = 1120$) is the total number of chunks, $q(.)$ is a quality function that maps bitrate to quality, $\delta_1 = 1$, and δ_2 and δ_3 are set to the maximum bitrate level in the bitrate list. To simplify the presentation of the QoE, we used a normalized QoE (N-QoE) with values between 0 and 1. To achieve that, we calculated the optimal achievable QoE (QoE*) offline for each bandwidth variation profile with the full knowledge of the network conditions using dynamic programming (N-QoE = QoE / QoE*).

The bandwidth prediction accuracy model is based on the Root Mean Square Error (RMSE) to compute the differences between the measured and predicted bandwidth values (sample standard deviation).

$$\text{RMSE} = \sqrt{\frac{1}{K}\sum_{i=1}^{K}\left(\frac{y_i - c_i}{c_i}\right)^2}, \quad \%\text{accuracy} = (1 - \text{RMSE}) \times 100 \quad (4)$$

4.1.5 Experimental Setup. Our setup consisted of two machines (running Ubuntu 18.04 LTS). One ran our dash.js player (see Figure 3) in a Google Chrome browser (v63) and one acted as a proxy to simplify bandwidth shaping between the client and server. The origin and edge servers were Cloudflare servers with CMAF packaging encoding and delivery enabled. Both machines were connected through a router and tc-NetEm was used to shape the network capacity according to our bandwidth variation profiles, including their RTTs and packet loss rates sequentially, as: (38 ms, 0.09%), (50 ms, 0.08%), (75 ms, 0.06%), (88 ms,0.09%), (100 ms,0.12%). We set the minimum and maximum buffer thresholds to half a second and the value of target latency, respectively. Furthermore, we used iPerf to generate random TCP-based cross traffic ranging from 0.5 to 2 Mbps. We compared ACTE against a set of dash.js-based ABR schemes with different configurations of bandwidth measurement algorithms. Additionally, we implemented the WSSL algorithm in the dash.js player. WSSL simultaneously downloads an earlier segment (that is fully available at the edge) and a current one while respecting the buffer occupancy and target latency. It then measures the bandwidth based on the redundant segment download. WSSL is an active probe-based technique that starts at every segment boundary and backs off immediately if the player encounters an increased latency (TCP-like congestion control). Table 3 lists the ABR schemes and bandwidth measurement algorithm combinations used in the experiments. For example, TH$_{sl}$ indicates that throughput-based (TH) ABR is combined with the SLBW algorithm.

Table 3: ABR and bandwidth measurement combinations.

Bandwidth Measurement	ABR Schemes		
	Throughput-based	Buffer-based	Hybrid
SLBW	TH$_{sl}$	-	-
EWMA	TH$_{ew}$	-	-
SWMA	TH$_{sw}$	BOLA$_{sw}$	Dynamic$_{sw}$
WSSL	TH$_{wss}$	-	-

4.2 Results

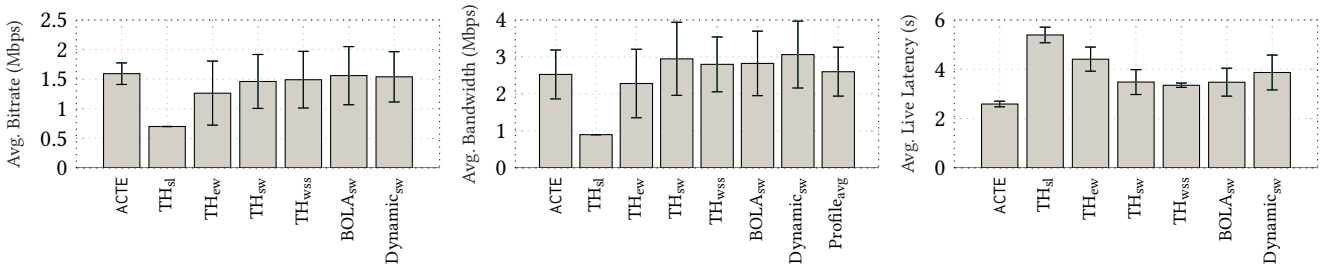
In this section, we show the average results for the bandwidth variation profiles. For accuracy, we repeated all experiments ten times for each ABR scheme with the same configuration, and all presented results are averages over the ten runs. Figure 6 illustrates the error bars (standard deviation) of the average selected bitrate, average measured bandwidth, and average live latency for different ABR schemes. Table 4 highlights the overall average results for various performance metrics. Two main observations can be drawn from these experiments. First, ACTE achieves the best results, outperforming other ABR schemes on each considered performance metric. TH$_{wss}$ is the runner-up, achieving the second best average N-QoE. Second, other ABR schemes suffer from poor performance because their bandwidth measurement algorithms may give incorrect values, leading to sub-optimal ABR decisions, unsatisfactory QoE and high latency.

Figure 6 and Table 4 show that ACTE selects the best bitrates resulting in a total average of 1.6 Mbps ranging from 1.4 to 1.7 Mbps (see Figure 6a), a smaller average number of switches (17), a lower average number of stalls (3) and a shorter stall duration of 0.76 seconds, a shorter average startup delay of 0.71 seconds, and thus, a higher average N-QoE of 0.95. This is because of the accurate bandwidth measurement and prediction employed by ACTE, where (*i*) the SWMA over the last three downloaded chunks outputs an accurate average bandwidth value (average of 2.6 Mbps), which closely approximates the average bandwidth profile values (see Figure 6b), and (*ii*) `ACTEUpdate()` (RLS-based) significantly improves the ABR selection for low-latency streaming by offering the bandwidth prediction for the next average bandwidth with a low average RMSE of 0.033 and a high average accuracy of 96.63%. All of these ACTE results contribute to achieving a low average latency of 2.5 seconds (see Figure 6c), which is close to the average buffer occupancy (see Table 4).

TH$_{wss}$ achieves the second best results (*e.g.*, N-QoE) after ACTE thanks to the TCP active probe-based technique that measures the bandwidth accurately by downloading a previous segment (using `xhr` API) available at the edge server, in parallel with the current chunk. However, this scheme incurs overhead (downloading redundant segments) that may harm the efficiency of TH$_{wss}$. For example, the average number of switches, average number of stalls and average stall duration reach 23, 16 and 9 seconds, respectively, and thus, the average N-QoE as well as live latency are negatively impacted. TH$_{sl}$ achieves the worst results in terms of average N-QoE and live latency. This is due to the fact that TH$_{sl}$ always plays the video at the lowest bitrate of 0.7 Mbps because the

Table 4: Average bitrate level, live latency, QoE with its metrics, RMSE and prediction accuracy.

	Avg. Selected Bitrate (Mbps)	Avg. Buffer Occupancy (s)	Avg. Live Latency (s)	Avg. # of Switches	Avg. # of Stalls & Duration (s)	Avg. Startup Delay (s)	Avg. N-QoE (N-QoE^{itu})	Avg. RMSE	Avg. Prediction Accuracy (%)
ACTE	1.4 to 1.7 (1.6)	2.1 to 3.0 (2.5)	2.3 to 3.0 (2.5)	17	3 & 0.76	0.71	0.95 (0.92)	0.033	96.63 %
TH$_{sl}$	0.7 to 0.7 (0.7)	3.6 to 5.0 (4.3)	5.0 to 5.8 (5.4)	0	2 & 0.86	1.46	0.42 (0.50)	-	-
TH$_{ew}$	0.6 to 1.9 (1.3)	1.9 to 3.9 (2.9)	4.0 to 5.0 (4.5)	18	21 & 66	1.06	0.60 (0.60)	-	-
TH$_{sw}$	1.0 to 1.9 (1.5)	1.9 to 3.5 (2.8)	3.0 to 4.0 (3.6)	24	27 & 33	1.03	0.76 (0.65)	-	-
TH$_{wss}$	1.0 to 2.0 (1.5)	2.0 to 3.1 (2.5)	3.1 to 3.3 (3.2)	23	16 & 9	0.88	0.85 (0.78)	-	-
BOLA$_{sw}$	1.1 to 2.1 (1.6)	1.6 to 3.0 (2.3)	3.0 to 4.0 (3.6)	20	58 & 119	1.66	0.65 (0.68)	-	-
Dymanic$_{sw}$	1.2 to 2.0 (1.5)	1.6 to 3.0 (2.4)	3.0 to 5.0 (3.9)	30	53 & 68	0.92	0.74 (0.72)	-	-



(a) Average selected bitrate. **(b) Average measured bandwidth.** **(c) Average live latency.**

Figure 6: Average selected bitrate, measured bandwidth and live latency for different ABR over 10 runs.

SLBW algorithm outputs inaccurate network bandwidth values where it always estimates the measured bandwidth to be equal to the segment bitrate (see Section 3.1), and thus, the player never switches up or down. Moreover, TH$_{sl}$ produces the highest average live latency of 5.4 seconds, ranging from 5.0 to 5.8 seconds (approximately equal to the segment duration).

As shown in Figure 6 and Table 4, TH$_{ew}$ attains inferior results, with an average N-QoE and live latency of 0.6 and 4.5 seconds, respectively. Also, TH$_{ew}$ experiences a high average number of stalls of 21 and a stall duration of 66 seconds. This is because the EWMA algorithm suffers from bandwidth measurement inaccuracy with many fluctuations as depicted in Figure 6b. Similarly, BOLA$_{sw}$ prefers to play the video at the highest possible quality (2 Mbps) most of the time, and hence, it experiences many (an average of 58) and long stalls (an average duration of 119 seconds). This happens because BOLA$_{sw}$ is a buffer-based ABR scheme that uses only the buffer occupancy to perform ABR decisions. In general, we can deduce that EWMA and buffer-based ABR schemes are not suitable in the context of low latency.

Results Summary. The percentages of improvement are computed by comparing ACTE's results with the achieved results of each scheme. The results are summarized in Table 5.

Table 5: Summary of the average results. Percentage improvements of ACTE's over the other schemes, at scale.

ACTE vs.	TH$_{sl}$	TH$_{ew}$	TH$_{sw}$	TH$_{wss}$	BOLA$_{sw}$	Dynamic$_{sw}$	Average
Avg. Selected Bitrate	128.6	23.1	6.7	6.7	0	6.7	28.6
Avg. Live Latency	53.7	44.4	30.6	21.9	30.6	35.9	36.2
Avg. # of Switches	N/A	5.6	29.2	26.1	15.0	43.3	23.8
Avg. # of Stalls	-50.0	85.7	88.9	81.3	94.8	94.3	65.8
Avg. Stall Duration	11.6	98.8	97.7	91.6	99.4	98.9	83.0
Avg. Startup Delay	51.4	33.0	31.1	19.3	57.2	22.8	35.8
Avg. N-QoE	126.2	58.3	25.0	11.8	46.2	28.4	49.3

5 CONCLUSIONS AND FUTURE DIRECTIONS

We proposed a bitrate adaptation scheme for chunked delivery in low-latency streaming applications. Our proposal, called ACTE, relies on three main components to perform ABR decisions: (1) a chunk-based sliding window moving average bandwidth measurement, (2) an RLS-based bandwidth prediction, and (3) a throughput-based bitrate selection logic. In this context, we found that the existing ABR schemes, which are based on segment-based bandwidth measurements, were not suitable for CMAF low-latency systems and suffered from many limitations that significantly impacted the user experience negatively. Results showed that ACTE achieved high performance by reducing the number of stalls by 65% and their durations by 83%, maintaining a low latency that ranged from 2.3 to 3 seconds (36% reduction), and providing 28% higher average bitrate, and 49% higher QoE compared to ABR schemes that used segment-based bandwidth measurements or EWMA. ACTE also achieved 96% accuracy in bandwidth prediction. In future work, we plan to extend the proposed solution by improving the bandwidth measurement and prediction further using deep reinforcement learning and an autoregressive integrated moving average (ARIMA) model, which we expect to be more effective under more dynamic network conditions. Also, we plan to investigate how new protocols such as HTTP/3 can be used efficiently for low latency.

ACKNOWLEDGMENTS

This research has been supported in part by the Singapore Ministry of Education Academic Research Fund Tier 1 under MOE's grant T1251RES1820 and the Austrian Research Promotion Agency (FFG) under the Next Generation Video Streaming project "PROMETHEUS."

REFERENCES

[1] 2018. *Information technology – Multimedia application format (MPEG-A) – Part 19: Common media application format (CMAF) for segmented media.* Standard ISO/IEC 23000-19:2018. International Organization for Standardization and International Electrotechnical Commission, Geneva, CH. https://www.iso.org/standard/71975.html

[2] ACTE Demo. 2019. [Online] Available: http://bit.do/ACTEdemo.

[3] A. C. Begen and Y. Syed. 2018. Are The Streaming format Wars Over?. In *2018 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*. 1–4. https://doi.org/10.1109/ICMEW.2018.8551563

[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 Communications Surveys & Tutorials* 21, 1 (2019), 562–585.

[5] N. Bouzakaria, C. Concolato, and J. Le Feuvre. 2014. Overhead and performance of low latency live streaming using MPEG-DASH. In *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*. 92–97. https://doi.org/10.1109/IISA.2014.6878732

[6] DASH Industry Forum (DASH-IF). 2019. dash.js JavaScript Reference Client. https://reference.dashif.org/dash.js/

[7] A. E. Essaili, T. Lohmar, and M. Ibrahim. 2018. Realization and Evaluation of an End-to-End Low Latency Live DASH System. In *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. 1–5. https://doi.org/10.1109/BMSB.2018.8436922

[8] Simon S. Haykin. 2008. *Adaptive filter theory*. Pearson Education India.

[9] P. Houzé, E. Mory, G. Texier, and G. Simon. 2016. Applicative-layer multipath for low-latency adaptive live streaming. In *2016 IEEE International Conference on Communications (ICC)*. 1–7. https://doi.org/10.1109/ICC.2016.7511550

[10] Will L. [n. d.]. Ultra-Low-Latency Streaming Using Chunked-Encoded and Chunked-Transferred CMAF. Akamai White paper. https://www.akamai.com/us/en/multimedia/documents/white-paper/low-latency-streaming-cmaf-whitepaper.pdf Online; accessed 10 January 2019.

[11] R. Pantos and W. May. 2017. *HTTP Live Streaming*. RFC 8216. RFC Editor. https://doi.org/10.17487/RFC8216

[12] W. Robitza, S. Göring, A. Raake, D. Lindegren, G. Heikkilä, J. Gustafsson, P. List, B. Feiten, U. Wüstenhagen, M.-N. Garcia, K. Yamagishi, and S. Broom. 2018. HTTP Adaptive Streaming QoE Estimation with ITU-T Rec. P. 1203: Open Databases and Software. In *Proceedings of the 9th ACM Multimedia Systems Conference (MMSys '18)*. ACM, New York, NY, USA, 466–471. https://doi.org/10.1145/3204949.3208124

[13] Y. Shuai and T. Herfet. 2018. Towards reduced latency in adaptive live streaming. In *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*. 1–4. https://doi.org/10.1109/CCNC.2018.8319262

[14] I. Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia* 18, 4 (April 2011), 62–67. https://doi.org/10.1109/MMUL.2011.71

[15] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. 1–9. https://doi.org/10.1109/INFOCOM.2016.7524428

[16] V. Swaminathan and S. Wei. 2011. Low latency live video streaming using HTTP chunked encoding. In *2011 IEEE 13th International Workshop on Multimedia Signal Processing*. 1–6. https://doi.org/10.1109/MMSP.2011.6093825

[17] J. Van Der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck. 2018. An HTTP/2 Push-Based Approach for Low-Latency Live Streaming with Super-Short Segments. *Journal of Network and Systems Management* 26, 1 (Jan. 2018), 51–78. https://doi.org/10.1007/s10922-017-9407-2

[18] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. *SIGCOMM Comput. Commun. Rev.* 45, 4 (Aug. 2015), 325–338. https://doi.org/10.1145/2829988.2787486