STALLION: Video Adaptation Algorithm for Low-Latency Video Streaming

Craig Gutterman[†], Brayn Fridman[†], Trey Gilliland[‡], Yusheng Hu[†], Gil Zussman[†]

[†]Electrical Engineering, Columbia University, New York, NY
[‡]Computer Science, Columbia University, New York, NY

ABSTRACT

As video traffic continues to dominate the Internet, interest in near-second low-latency streaming has increased. Existing low-latency streaming platforms rely on using tens of seconds of video in the buffer to offer a seamless experience. Striving for near-second latency requires the receiver to make quick decisions regarding the download bitrate and the playback speed. To cope with the challenges, we design a new adaptive bitrate (ABR) scheme, *Stallion*, for **STA**ndard **L**ow-**LA**tency v**I**deo c**ON**trol. *Stallion* uses a sliding window to measure the mean and standard deviation of both the bandwidth and latency. We evaluate Stallion and compare it to the standard DASH DYNAMIC algorithm over a variety of networking conditions. *Stallion* shows 1.8x increase in bitrate, and 4.3x reduction in the number of stalls.

CCS CONCEPTS

• **Information systems → Multimedia streaming**;

KEYWORDS

HTTP adaptive streaming, DASH, low-latency

ACM Reference Format:
Craig Gutterman, Brayn Fridman, Trey Gilliland, Yusheng Hu, Gil Zussman. 2020. STALLION: Video Adaptation Algorithm for Low-Latency Video Streaming. In *11th ACM Multimedia Systems Conference (MMSys'20), June 8–11, 2020, Istanbul, Turkey.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3339825.3397044

1 INTRODUCTION

Video has dominated Internet traffic in recent years. Recently, there has been an increased usage of *live video streaming*, and it is expected to grow to 13% of total Internet traffic by 2021 [3]. Due to the spread of COVID-19 and encouragement to "shelter in place", there has been an increase in live streaming of over 10% from Q4 2019 to Q1 2020 for Twitch, Facebook Gaming, and YouTube Gaming [6]. To offer clients a high quality of experience, *it is essential for live streaming services to minimize the video streaming live latency.* Live latency is defined here as the time between when the video was captured to the time it is displayed on the user device. On-demand streaming services usually rely on tens of seconds of video to be stored on the client buffer. This gives the service time to adapt to changes in network conditions while still offering a good quality of experience (QoE). On the contrary, live streaming requires much lower live latency, resulting in a reduced buffer on the client device. The service needs to *adapt quickly to network changes while also offering a high video resolution and minimizing the number of stall events.*

A majority of streaming services rely on HTTP Adaptive Streaming (HAS) to deliver video over the Internet. One of the most popular techniques is Dynamic Adaptive Streaming over HTTP (DASH) or MPEG-DASH [23, 25]. For HAS, each video is divided into shorter segments of video. The segment length is the number of seconds of video for each segment and is usually a few seconds of video. Each of the segments is encoded in multiple resolutions and bitrates. The client player uses an adaptive bitrate (ABR) algorithm to dynamically request segments with the objective of maximizing the QoE. DASH ABR algorithms are either buffer-based (e.g., BOLA [22]), throughput-based (e.g., Oboe [8]) or hybrid (e.g., DYNAMIC [21]). One of the main issues with these algorithms is that they are not designed to work for low-latency streaming.

For near-second low-latency streaming, there is only a second or two of video buffered on the user device. Therefore, the client player needs to ensure the timely delivery of the next video segment. Unstable network conditions can result in fluctuations to the throughput and the latency (time between a request and response). Unexpected reductions in the throughput or an increase in the latency will add delay and could result in the video stalling.

Prior algorithms incorporated into DASH only rely on using the buffer level as well as the mean (either sliding window or exponential weighted moving average (EWMA)) of throughput and latency. These statistics are insufficient for dealing with fluctuations of throughput and latency during playback. To deal with variable network conditions, near-second streaming requires additional information about the variations in the throughput and latency on a per segment basis.

Therefore, we propose an ABR algorithm that incorporates standard deviation of the network measures. We present *Stallion*, for **STA**ndard **L**ow-**LA**tency v**I**deo c**ON**trol. We design a throughput-based system that incorporates the standard deviation of these throughput and latency measures. *Stallion* uses a sliding window to measure the mean and standard deviation of both the bandwidth and latency.

We implement *Stallion* in the modified dash.js reference player provided by the ACM MMSys'20 Grand Challenge [4]. This player is modified from the standard DASH player [2] to pre-request low latency segments. This enables the player to request the segments before they are available on the server.

To evaluate the performance of *Stallion*, we compared it against DYNAMIC [21] with the 5 network profiles provided by [4]. Our results show that *Stallion* outperforms DYNAMIC. In particular, it achieves higher QoE with a 1.8x increase in bitrate, and 4.3x reduction in the number of stalls. The code for *Stallion* is available for testing by the reviewers at [7].

The rest of this paper is organized as follows. Section 2 provides background on ABR algorithms. Section 3 presents our preliminary findings on low-latency streaming. It then continues to describe the system and implementation of *Stallion*. Section 4 describes our experimental setup. It then discusses the QoE evaluation and the network traces used for evaluation. In addition, we compare the performance of *Stallion* and DYNAMIC. Section 5 concludes and discusses future directions.

2 BACKGROUND AND RELATED WORK

HTTP Adaptive Streaming (HAS) is being used by a majority of streaming services to deliver video over the internet. One of the most common formats is Dynamic Adaptive Streaming over HTTP (DASH) or MPEG-DASH [23, 25].

HAS encodes a video in multiple resolutions and bitrates. Each encoding is then divided into a number of *segments* of variable length (usually a few seconds of video) [17]. Clips are either encoded with Variable Bitrate (VBR) encoding or with Constant Bitrate (CBR).

Details that the client can download from local servers are given in a DASH Media Presentation Description (MPD) file. For each segment request, an HTTP GET request is sent to the server. In response to the GET request from the client the server sends the requested segment. This allows the video player of the client to dynamically request segments based on the ABR algorithm. The client maintains a buffer to temporarily store segments of the video. This is used to ensure a smooth playback. A stall event occurs when no segments are stored in the buffer.

While a majority of services use HAS, each has its own method of ABR streaming [17]. ABR requires either TCP or another reliable transport [12] and is usually delivered over HTTP(S). The ABR algorithms rely on statistics such as throughput, buffer health, or a hybrid (throughput and buffer) to determine the optimal segment to request. Throughput is measured for each segment to estimate bandwidth. Thus, the player can switch to a segment with a lower bitrate when the estimated bandwidth or buffer is low to avoid stalling. If the bandwidth becomes available at a future time, the player can switch back to a higher bitrate to provide a better user experience. Below we provide further details about the different types of ABR algorithms.

Buffer-based: Buffer-based ABR algorithms rely on using the buffer level to determine the optimal bitrate of the requested segment. BOLA [22] is an ABR algorithm that uses buffer occupancy to designate the bitrate of the next segment. As the buffer occupancy grows, higher bitrate segments are preferred. One drawback of BOLA is that it usually performs better with higher buffer levels [21]. BBA [14] selects the bitrate with the goal of minimizing stalls and keeping the buffer occupancy level above a certain threshold. When the level exceeds 15 sec, it switches to the highest available bitrate.

Throughput-based: Throughput-based ABR algorithms collect and calculate all the transmission information between the server and the client to estimate the available throughput. The estimated throughput is then used to determine the optimal bitrate for the next video segment. Oboe [8] aims to ensure good user QoE by enabling auto tuning configuration parameters of the ABR algorithm based on the current network state. Additional throughput-based algorithms include FESTIVE [15] and PANDA [16].

Hybrid-based: Hybrid-based ABR algorithms combine both buffer-based and throughput-based approaches. DYNAMIC [21] is a hybrid algorithm that is built into the official DASH reference player. It uses a version of BOLA for its buffer-based algorithm (when the buffer is high) as well as a throughput rule for its throughput-based algorithm (when the buffer is low or empty). Additional hybrid based algorithms include ELASTIC [11] and MPC [26]. Penseive [18] utilized reinforcement learning for generating optimal ABR algorithms. The neural network model is based on the data from client players. Penseive gradually learns to make better ABR decisions through reinforcement in the form of reward signals that reflect video QoE for past decisions. HotDASH [20] also uses reinforcement learning for its decision engine as well as prefetching of video segments.

Recently there has been additional research focusing on ABR algorithms specifically designed to deal with low-latency streaming. LOLYPOP [19] is designed for low-delay video streaming and uses TCP throughput predictions on times scales of 1 to 10 sec. ACTE [9] uses a sliding window to estimate bandwidth and uses a linear adaptive filter to predict future bandwidth.

Content providers are not the only companies who have a stake in the live video streaming sector. Extensive research efforts have been dedicated to monitoring and inferring QoE related video statistics from encrypted video traffic [13, 24]. This information could allow service providers to prioritize clients to improve their video QoE [10].

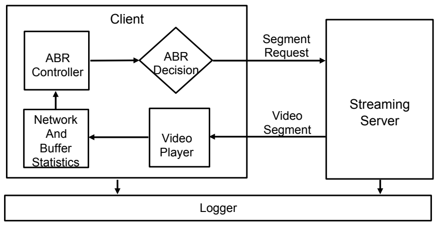
3 STALLION SYSTEM DESIGN

The goal of ACM MMSys'20 Grand Challenge is to use the dash.js reference player to create an ABR algorithm for near-second low-latency live streams. We were provided the dash.js reference player modified to pre-request low latency segements [4]. The low latency server delivers segments with length of 0.5 sec.

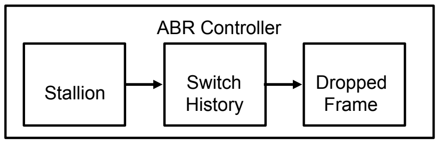
We use this player to observe the performance of DYNAMIC in low latency scenarios. We then develop *Stallion*, a throughput-based ABR algorithm that is designed to operate effectively in low-latency settings with unstable network conditions.

3.1 Preliminary Low-Latency DASH Findings

We use the dash.js player provided in [4] to test the performance of ABR algorithms. Additionally, we implemented an automated system where we can log the performance of the player, and analyze its performance. The system diagram in Fig. 1(a) shows the major components used for testing. The ABR controller uses statistics from the network and buffer level to determine the bitrate of the next segment.



(a)



(b)

Figure 1: (a) The dash.js player given with modifications to switch ABR algorithm and log results, (b) *Stallion* ABR controller containing the *Stallion*, Swith History and Dropped Frame rules.

The default ABR algorithm for DASH is DYNAMIC [21]. This controller contains the DYNAMIC which has a collection of ABR rules including ThroughputRule, BolaRule, InsufficientBufferRule, SwitchHistoryRule, and DroppedFramesRule. The hybrid model dynamically chooses either the BOLA or Throughput rule depending on the buffer level. The three remaining rules are also used to offer an option of quality requested. The lowest of the 4 qualities is the final chosen quality.

The network profiles provided by [4] and described in Section 4.2 are used to test the initial performance. One of the preliminary findings is that if the buffer is low, the throughput rule is usually chosen instead of BOLA. In addition, the ultimate decision is usually the result of the insufficient buffer rule being chosen because the buffer is usually less than 0.5 sec.

Even with the insufficient buffer rule being chosen, there were frequent stalls as there was not sufficient time to adjust to changes in the network conditions. The stalls led to an increase in the live playback latency. The playback rate is increased when the live-latency delay is greater than the target latency. In an effort to minimize the live latency delay, the playback controller increases the playback rate. Increasing the playback rate when the buffer is already low results in additional stalls. The DYNAMIC rules had a difficult time estimating the available bandwidth and therefore the segment bitrate was lower than the optimal bitrate supported by the network conditions.

3.2 Stallion Implementation

The constraints that were given by the Grand Challenge were that the segment duration, the segment chunk size, and the segment request behavior could not be changed. The observations in 3.1 influenced the design of *Stallion*. Beyond the design of a custom rule Stallion.js, we modified additional functionality of ThroughputHistory.js, PlaybackController.js, AbrController.js, and index.html. The resulting ABR controller for *Stallion* is represented in Fig. 1(b). In addition to the *Stallion* rule, we use the Switch History and Dropped Frame rules that were also used in DYNAMIC.

The first parameter that needs to be initially set is the target latency of the ABR algorithm. The initial setting of target latency of the near-second low-latency DASH player is 1 sec. After 1 segment (0.5 sec) of playback, the next segment should be sent to the client. Consequently, the next segment needs to be downloaded before the current one is playing or otherwise a stall will occur. This gives a minimal amount of time to deal with any unstable network conditions. Therefore, we update the initial target latency of the DASH player to 1.5 sec. This leaves 1 additional segment in the buffer of the client and additional time for the DASH player to adapt to network conditions.

Based on our observations, we noted issues with the video playback rate control. When the playback falls behind the target latency, the controller attempts to catch up to the target latency by increasing the playback rate. This occurs even when the buffer is low (but not completely stalled). Increasing the playback rate when the buffer is low would likely only increase the number of stalls, thereby increasing the latency further. We make the following modification to improve the performance for low latency. When the player has a latency greater than the target latency, the playback rate will only increase if the buffer level is greater than 0.6 sec.

Unstable network conditions result in client bandwidth fluctuations during playback. Relying only on the mean of the throughput and latency leaves out details in the statistics of the network pattern that can be beneficial when determining the optimal video segment to request. Therefore, we develop a custom throughput-based rule, *Stallion*, that relies on the use of the standard deviations of both latency and throughput to make a safe estimate for our bitrate selection. Fig. 1(b) shows the Stallion ABR controller containing the Stallion rule, along with the Switch History and Dropped Frame rules.

The throughput, Thr^s, of sample s is measured by taking the download size and dividing it by the download time. Each latency sample, L^s, is measured by the time the request is sent to the time the first packet of the segment is delivered. The first data points needed are the average throughput, $\bar{Thr}_n$, and the average latency, $\bar{L}_n$. These metrics are already built in ThroughputHistory.js. The sliding window sample mean is calculated over a maximum of $n = 10$ samples.

In addition, we add methods to ThroughputHistory.js (part of the Network and Buffer Statistics in Fig. 1(a)) to calculate the sample standard deviation of both the throughput and latency for the sliding window. The sample standard deviation of the throughput is represented as σ_n^{Thr}. The sample standard deviation of the latency is represented as σ_n^L. These data points are used to assign safe estimates for the bitrate and latency. Accordingly, we set:

$$\hat{Bitrate} = \bar{Thr}_n - z_{Thr}\sigma_n^{Thr}$$

$$\hat{Latency} = \bar{Thr}_n + z_L\sigma_n^L$$

These two estimates are then used in abrController.getQualityForBitrate() to determine the highest possible segment bitrate that can be requested given the constraints of

$\hat{Bitrate}$ and $\hat{Latency}$. The goal of this function is determine the maximal bitrate of the next segment that can be delivered during a segment duration. The latency is subtracted from the fragment duration to determine the dead time. The remaining time is the amount of time during the segment duration that can be used to download. This time is then multiplied with $\hat{Bitrate}$ to determine the actual realizable bitrate. The segment with the highest bitrate less than the realizable bitrate is the bitrate chosen by *Stallion*.

The bitrate chosen by the *Stallion* rule is then accompanied by the results of the Switch History and Dropped Frames rules. The final quality is chosen by finding the minimum quality of active rules. The complete code for our implementation can be found at [7].

4 EVALUATION

We develop a testing and logging framework to measure the performance of the proposed ABR algorithm vs. a baseline system. The evaluation of the algorithms is done by using a variety of network profiles with unstable network conditions. We describe the relevant metrics used for evaluation along with the calculation of the QoE metric. Finally, we use these components to compare the performance of *Stallion* and DYNAMIC.

4.1 Experimental Setup

As ABR streaming is comprised of many components working together to optimize playback quality, each component relies on observations about the current network conditions available to the client. To measure the performance and provide us with feedback on how ABR modifications impact the playback experience, we write various metrics throughout the framework to a log file. With this log file, we measure the performance of each ABR algorithm on the playback quality. The testing script runs through multiple runs of each network trace. Further details about the network traces are described in Sec. 4.2.

To sufficiently test our modifications, we develop an automated script to run through multiple tests of our modified player on each network testing profile. This script saves the test log file and results JSON file from each individual test. They are then used to measure the performance throughout the entire experiment.

The video used for testing is Big Buck Bunny [1]. Each video is encoded in 3 bitrates of 200 Kbps, 600 Kbps, and 1000 Kbps. The associated video qualities of these bitrates are 360p, 480p, and 720p, respectively.

All experiments were conducted on an Apple Macbook Pro laptop with a 2.8 GhZ Intel Core I7 with 16 GB of RAM. Both the server and the dash player were run on the laptop. The dash.js player ran in Google Chrome browser.

4.2 Network Traces

There are 5 network profiles which are used for testing. The details of the 5 networking patterns can be seen in Table 1. Each network profile has a network pattern that is repeated once per experiment.

The first network profile is Cascade. This pattern lasts for roughly 150 seconds with 30 second intervals at network data rates of 1200 Kbps, 800 Kbps, 400 Kbps, 800 Kbps, and back to 1200 Kbps. The second network profile is Intra Cascade. This profile contains a pattern with 9 steps of 200 Kbps for 15 seconds each. The steps begin at 1000 Kbps going down to 200 Kbps, and then back up to 1000 Kbps. The third profile is labeled as Spike. This profile has 3 steps lasting 10 sec each: 1200 Kbps, 300 Kbps, and 800 Kbps. The fourth profile has a pattern that is 30 seconds and switches between 500 Kbps and 1200 Kbps every 5 sec. The fifth profile tested is Fast Jitters, with a network pattern of 11.6 seconds. The data rates of this profile are 500 Kbps, 1200 Kbps, 500 Kbps, 1200 Kbps, 500 Kbps, and 1200 Kbps with durations of 0.25 sec, 5 sec, 0.1 sec, 1 sec, 0.25 sec, and 5 sec, respectively.

Table 1: Network profiles from ACM MMSys'20 Grand Challenge

Notation	Time Duration (sec)	Data Rate (Kbps)
Cascade	(30,30,30,30,30)	(1200,800,400, 800,1200)
Intra Cascade	(15,15,15,15,15,15,15,15,15)	(100,800,600,400, 200,400,600,800,1000)
Spike	(10,10,10)	(1200,300,800)
Slow Jitters	(5,5,5,5,5,5)	(500,1200,500,1200, 500,1200)
Fast Jitters	(0.25,5,0.1,1,0.25,5)	(500,1200,500,1200, 500,1200)

Table 2: Parameters for QoE function

Notation	Meaning
s	A segment
R_s	Reward of Segment bitrate (Kbps)
E	REbuffering time (seconds)
L	Live latency (seconds)
P	Playback Speed
S	Total number of segments
α	Bitrate reward factor (0.5 sec)
β	Rebuffering penalty factor (=R_s for 1000 Kbps)
γ	Live Latency penalty factor (if L ≤ 1.1 sec then $= 0.005$, otherwise $= 0.01$)
σ	Playback speed penalty factor (=R_s for 200 Kbps)
μ	Bitrate switch penalty factor (0.02 sec)

4.3 QoE Model

We focus on 5 metrics for our evaluation criteria. The first criteria is the average selected bitrate (Kbps). The second metric is average live latency. The live latency is measured as the amount of time between when the original video was 'created' until it is played back. The third criteria is the average number of bitrate switches. While higher bitrates are in general better for the QoE of the user, bitrate switches have a negative impact on the user's viewing experience. The fifth metric is stall duration. The longer the stall duration, the higher the negative impact on the user's experience.

The QoE model used for evaluation is based on the document provided by [5]. We use the per-segment QoE for evaluation. For the low latency model, segment duration is 0.5 sec. A list of notations for QoE function can be found in Table 2. Overall, the QoE model is calculated as:

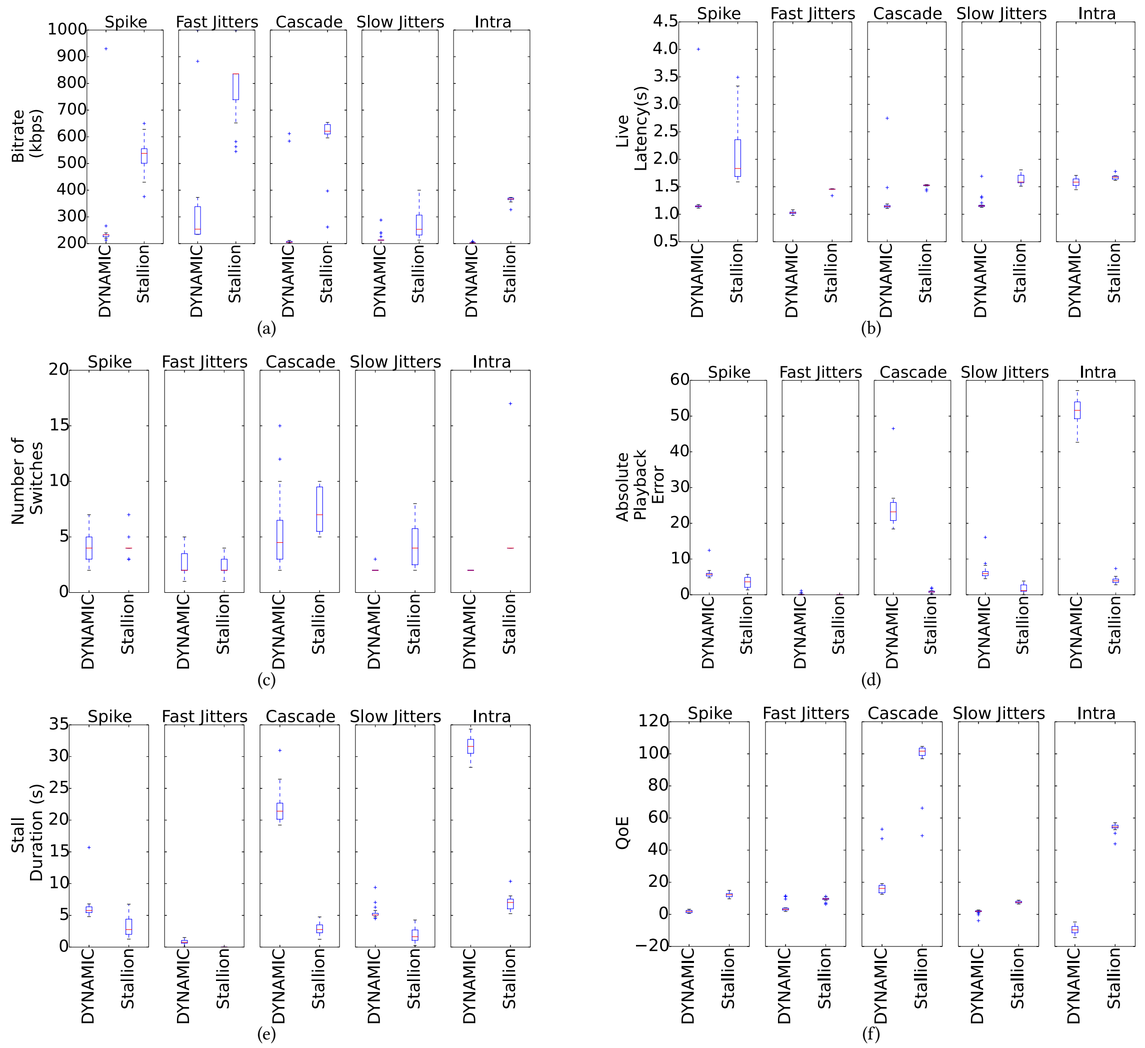


Figure 2: Performance comparison of DYNAMIC and *Stallion* for each network profile: (a) Average bitrate, (b) Average live latency, (c) Number of playback rate switches, (d) Absolute Playback Error, (e) Stall Duration(sec), (f) QoE. For each box plot, the middle red line is the median value. The bottom of the box represents Q1 (25-percentile) and the top of the box represents Q3 (75-percentile) of the dataset, respectively. The lower extended line represents Q1-1.5IQR, where IQR is the inner quartile range (Q3-Q1). The higher extended line represents Q3 + 1.5IQR.

$$QoE = \sum_{s=1}^{S}(\alpha R_s - \beta E_s - \gamma L_s - \sigma|1 - P_s|) - \sum_{s=1}^{S-1}\mu|R_{s+1} - R_s|$$

To make all the rewards on the same scale we set the reward for specific bitrates (Kbps) as:

$$R_s = log_{10}(\frac{\text{Bitrate}}{100})$$

4.4 Results

We evaluate the low-latency performance of *Stallion* against the DYNAMIC ABR controller. We tune the parameters of z_{Thr} and z_L to optimize the performance of *Stallion* . The final rule tested in the paper has z_{Thr} set to 1 and z_L set to 1.25. Each ABR algorithm was run 20 times on each network profile. Fig. 2 shows the results of these tests. We further discuss the results for each QoE metric and the total QoE as follows:

Bitrate: One of the primary objectives of an ABR algorithm is to maximize the bitrate of the video playing. For all 5 network profiles, DYNAMIC has a median bitrate less than 300 Kbps. The median bitrate for *Stallion* is greater for every network profile tested. In 4 of the 5 profiles, there was at least a 150 Kbps increase in average bitrate. Overall, the average bitrate increased from 290 Kbps to 530 Kbps, a 1.8x increase in average bitrate.

Average live latency: For DYNAMIC, the target latency is set to 1 sec, while it is set to 1.5 for *Stallion*. The average live latency for DYNAMIC is less than 1.5 sec in all network profiles except Intra. The worst performing network profile for *Stallion* is Spike. The median live latency is 1.8 sec for Spike, while the rest of the network profiles are slightly above 1.5 sec.

Number of playback rate switches: We analyze the stability of the playback based on the number of playback rate switches. The performance for both ABR algorithms tested is approximately the same for Spike and Fast Jitters. There are approximately 2 additional switches in *Stallion* compared to DYNAMIC for Cascade, Slow Jitters, and Intra. Each ABR algorithm has a median number of switches less than 10 for each network profile.

Absolute Playback Error: The absolute playback error for each segment is calculated as $|1 - P_s|$. The absolute playback error is calculated as the sum of the playback error for each video segment of an experiment. Both *Stallion* and DYNAMIC perform extremely well for Fast Jitters, as the playback rate is usually kept constant at 1. For *Stallion*, the absolute playback error is kept below 5 for all experiments with a median of 1.37. DYNAMIC performs considerably worse for the Cascade and Intra network profiles.

Stall Duration: One of the main issues that can occur during near-second low-latency playback is the number of stall events and the length of time the video is stalled. DYNAMIC has a median of 6 sec of stall duration while *Stallion* has a median of 3 sec for the Spike profile. For Fast Jitters, both perform well with less than 1 second of stall duration. For Cascade, there is a large improvement in stall duration, as Dynamic has a median of 22 sec, while *Stallion* has a median of 3 sec. DYNAMIC has a stall duration of 5 sec, while *Stallion* has a stall duration of under 2 sec for the Slow Jitters network profile. DYNAMIC performs the worst for the Intra network profile. It has a median stall duration over 30 sec. This network profile is also the worst for *Stallion* with a median stall duration of 7 sec. Overall ,the average stall duration of *Stallion* is 3.1 sec, compared to an average stall duration of 13.3 sec for DYNAMIC. This is a 4.3x improvement in average stall duration for *Stallion*.

QoE: The QoE score is calculated based on the details given in 4.3 and Table 2. The weights given can be adapted depending on the objectives. *Stallion* outperformed the QoE in all network profiles. The profiles that saw the biggest improvement were Cascade and Intra.

Overall, *Stallion* has improved QoE for near-second low-latency streaming. The design of *Stallion* that uses the information provided by the simple second order metric (standard deviation) of the latency and bitrate estimates allow it to make safer decisions. These safer decisions allow *Stallion* not only to reduce the number of stalls considerably, but also to improve the bitrate without significantly increasing the number of bitrate changes.

5 CONCLUSION

We present *Stallion*, an ABR controller designed for near-second low-latency video control for the ACM MMSys'20 Grand Challenge. The code can be found at [7]. *Stallion* is simple to implement and relies on (i) a segment based sliding window of the bitrate and latency, and (ii) the mean and standard deviation of this window to estimate the bitrate and latency available for the next segment. We determine that specifically with a near-second target latency, using the standard deviation along with the mean gives a safer prediction for available bandwidth and thus reduces the likelihood of stalling.

Stallion is evaluated against the DASH standard DYNAMIC over a variety of networking conditions. *Stallion* shows 1.8x increase in bitrate, and a 4.3x reduction in the number of stalls.

For future research we would like to add additional network profiles and to compare additional ABR algorithms. In addition, we would like to add a machine learning based system that auto adapts the tuning parameters of *Stallion* during video playback.

ACKNOWLEDGMENTS

This work was supported in part by NSF grants CNS-1910757.

REFERENCES

[1] Big buck bunny. https://peach.blender.org/.

[2] Dash industry forum (dash-if). https://reference.dashif.org/dash.js/v3.0.3/samples/dash-if-reference-player/index.html.

[3] Cable live video usage will increase 15-fold by 2021, cisco predicts. https://www.fiercevideo.com/cable/live-video-usage-will-increase-15-fold-by-2021-cisco-predicts, 2017.

[4] ACM MMSys 2020 grand challenge. https://github.com/twitchtv/acm-mmsys-2020-grand-challenge, 2020.

[5] Nqoe. https://github.com/twitchtv/acm-mmsys-2020-grand-challenge/blob/master/NQoE.pdf, 2020.

[6] Twitch breaks various viewership records amid coronavirus quarantine. https://www.hollywoodreporter.com/news/twitch-breaks-viewership-records-coronavirus-quarantine-1287894, 2020.

[7] Twitch challenge. https://github.com/Wimnet/twitch_challenge, 2020.

[8] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett, B. Ribeiro, J. Zhan, and H. Zhang. Oboe: auto-tuning video abr algorithms to network conditions. In *Proc. ACM SIGCOMM*, pages 44–58, 2018.

[9] A. Bentaleb, C. Timmerer, A. C. Begen, and R. Zimmermann. 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*, 2019.

[10] R. Bhattacharyya, A. Bura, D. Rengarajan, M. Rumuly, S. Shakkottai, D. Kalathil, R. K. Mok, and A. Dhamdhere. Qflow: A reinforcement learning approach to high qoe video streaming over wireless networks. In *Proc. ACM MobiHoc*, 2019.

[11] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. Elastic: a client-side controller for dynamic adaptive streaming over HTTP (DSAH). In *2013 20th International Packet Video Workshop*. IEEE, 2013.

[12] R. T. Fielding and J. F. Reschke. Hypertext transfer protocol (HTTP/1.1): message syntax and routing. *RFC*, 7230:1–89, 2014.

[13] C. Gutterman, K. Guo, S. Arora, X. Wang, L. Wu, E. Katz-Bassett, and G. Zussman. Requet: Real-time QoE detection for encrypted YouTube traffic. In *Proc. ACM MMSys*, 2019.

[14] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proc. ACM SIGCOMM*, 2014.

[15] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proc. ACM CONEXT*, 2012.

[16] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. Probe and adapt: Rate adaptation for http video streaming at scale. *IEEE JSAC*, 32(4):719–733, 2014.

[17] A. Mansy, M. H. Ammar, J. Chandrashekar, and A. Sheth. Characterizing client behavior of commercial mobile video streaming services. In *Proc. ACM MoVid*, 2014.

[18] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive video streaming with pensieve. In *Proc. ACM SIGCOMM*, 2017.

[19] K. Miller, A.-K. Al-Tamimi, and A. Wolisz. Qoe-based low-delay live streaming using throughput predictions. *ACM TOMM*, 13(1):1–24, 2016.

[20] S. Sengupta, N. Ganguly, S. Chakraborty, and P. De. Hotdash: Hotspot aware adaptive video streaming using deep reinforcement learning. In *Proc. IEEE ICNP*, 2018.

[21] K. Spiteri, R. Sitaraman, and D. Sparacio. From theory to practice: Improving bitrate adaptation in the dash reference player. *ACM TOMM*, 15(2s):1–29, 2019.

[22] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *Proc. IEEE INFOCOM*, 2016.

[23] T. Stockhammer. Dynamic adaptive streaming over HTTP -: standards and design principles. In *Proc. ACM MMSys*, 2011.

[24] S. Wassermann, M. Seufert, P. Casas, L. Gang, and K. Li. Let me decrypt your beauty: Real-time prediction of video resolution and bitrate for encrypted video streaming. In *Proc. IEEE TMA*, 2019.

[25] N. Weil. The state of MPEG-DASH 2016. http://www.streamingmedia.com/Articles/Articles/Editorial/Featured-Articles/The-State-of-MPEG-DASH-2016-110099.aspx.

[26] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proc. ACM SIGCOMM*, 2015.