Evaluation of CMAF in Live Streaming Scenarios

Tomasz Lyko
Lancaster University
United Kingdom
t.lyko@lancaster.ac.uk

Matthew Broadbent
Lancaster University
United Kingdom
m.broadbent@lancaster.ac.uk

Nicholas Race
Lancaster University
United Kingdom
n.race@lancaster.ac.uk

Mike Nilsson
British Telecommunications
United Kingdom
mike.nilsson@bt.com

Paul Farrow
British Telecommunications
United Kingdom
paul.farrow@bt.com

Steve Appleby
British Telecommunications
United Kingdom
steve.appleby@bt.com

ABSTRACT

HTTP Adaptive Streaming (HAS) technologies such as MPEG DASH are now used extensively to deliver television services to large numbers of viewers. In HAS, the client requests segments of content using HTTP, with an ABR algorithm selecting the quality at which to request each segment to trade-off video quality with the avoidance of stalling. This introduces significant end to end latency compared to traditional broadcast, due to the the client requiring a large enough buffer for the ABR algorithm to react to changes in network conditions in a timely manner. The recently standardised Common Media Application Format (CMAF) has helped address the issue of latency by defining segments as composed of independently transferable chunks. In this paper, we describe a simulation model we have developed to evaluate the performance of four popular ABR algorithms using DASH and CMAF in various low latency live streaming scenarios. Realistic network conditions are used for the evaluation, which are based on throughput data taken from the CDN logs of a commercial live TV service. We quantify the performance of the ABR algorithms using a selection of QoE metrics, and show that CMAF can significantly improve ABR performance in low delay scenarios.

CCS CONCEPTS

• **Information systems → Multimedia streaming**.

KEYWORDS

CMAF, DASH, ABR, live, latency, video streaming, adaptive streaming

ACM Reference Format:
Tomasz Lyko, Matthew Broadbent, Nicholas Race, Mike Nilsson, Paul Farrow, and Steve Appleby. 2020. Evaluation of CMAF in Live Streaming Scenarios. In *Istanbul'20: ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video, June 10-11, 2020, Istanbul, Turkey*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

NOSSDAV'20, June 8-11, 2020, Istanbul, Turkey

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

1 INTRODUCTION

Video streaming is the dominant and fastest growing source of traffic on the Internet[1]. A large proportion of this traffic is delivered by HTTP Adaptive Streaming (HAS) technologies such as MPEG Dynamic Adaptive Streaming over HTTP (DASH)[12], where content is split into short segments (usually from 2-10 seconds), encoded at multiple bit rates and hosted on a standard HTTP server. A manifest file is created that indicates the encoded bit rates and where the content can be obtained. The client requests the manifest file, then makes HTTP requests for consecutive segments of content at bit rates selected by an Adaptive Bit Rate (ABR) algorithm to maximise the Quality of Experience (QoE) of the viewer.

The use of HAS technologies such as DASH usually causes high end to end latency compared to traditional terrestrial or satellite broadcast television due to buffering of received but not yet decoded segments in the player, this provides time for the player to adapt to changes in the network throughput by switching to an appropriate video quality before the buffer is depleted and play-out stalls. For live streaming scenarios, the potential maximum buffer level is limited by how far behind the live edge the player is.

The Common Media Application Format (CMAF) has been standardised [16] to help address the latency issue by enabling a segment to be created as a sequence of chunks, each of which can be delivered to the client as soon as it is created, rather than having to wait until a whole segment has been created as with DASH.

In this paper, we describe a simulation model we have developed to evaluate the performance of four popular ABR algorithms when used in a low latency live streaming environment with realistic network conditions, where the clients run MPEG DASH with and without the use of CMAF chunks.

2 BACKGROUND AND RELATED WORK

2.1 Latency

Shuai et al. [23] found, that the main contributor to latency in HAS is the client buffer, which stores fetched video segments that are queued for playback. Its size is determined by an ABR algorithm's ability to adapt to changing network conditions in a timely manner. It needs to be large enough to give the ABR algorithm enough time to measure network conditions and change video quality before any rebuffering occurs. Hence, an insufficient buffer size will lead to a high level of rebuffering events.

[1] Cisco Visual Networking Index: Forecast and Trends, 2017-2022

Lohmar et al. [20] outlined four sources of delay that are specific to HTTP streaming. The main one is the client buffer, and the other three are as follows. Asynchronous fetching of media segments, where a client may issue an HTTP GET request for a segment some time after it is made available. HTTP download time, where segment size and available bandwidth determine how fast a segment can be fetched. Segmentation delay, which introduces a delay of at least one segment duration, but which CMAF could reduce to one chunk duration as segments can be divided into chunks [16].

2.2 Common Media Application Format

The Common Media Application Format (CMAF) [16] allows a segment to be created as a sequence of chunks. Whereas a DASH segment must be completely written to a server before it can be addressable and requested, a segment with CMAF chunks can be requested as soon as the first chunk is written to the server. This reduces the minimum latency in live streaming from one segment duration to one chunk duration, although it is still only possible to change the video quality and video bit rate at segment boundaries.

A segment containing CMAF chunks can be requested as soon as the first chunk is created. HTTP/1.1 Chunked Transfer enables subsequent chunks of a segment to be delivered as soon as they become available without additional requests from the player.

Figure 1 demonstrates the difference between segment delivery in DASH with and without CMAF when the player is close to the live edge. DASH segments can be requested as soon they become available, periodically every segment duration, whereas segments with CMAF chunks can be requested as soon their first chunk becomes available. The first chunk is delivered immediately, and the remaining chunks are delivered as they become available every chunk duration. In the illustrated case, where segments have four CMAF chunks, three CMAF chunks could have been delivered before the DASH segment becomes available.

However, since chunk delivery at the live edge is restricted by the encoder, estimation of network throughput becomes difficult for applications that have no direct visibility of any idle periods between chunks. Bentaleb et al. [14] attempted to solve this problem by ignoring throughput measurements for chunks that contain the idle time in their download times. The simulation model that we have developed calculates the delivery time of chunks correctly, and hence we do not consider this issue further.

2.3 ABR Algorithms

Kua et al. [18] published a survey of ABR algorithms, including **Panda**[19], **Festive**[17], **MPC**[25], and **Bola**[24]. They stated that the goals of ABR algorithms are to maximize the average video quality, minimize the rebuffer rate, the frequency of changes of video quality and the start-up delay; they also state that trade-offs are required, as these goals are in competition with each other.

Panda uses a probe-and-adapt approach similar to TCP's congestion control. It determines a target average data rate, based on which an appropriate quality bitrate is selected. It monitors throughput and adjusts the target average data rate accordingly, as well as calculating the inter-request time for each segment, allowing the buffer level to move towards the configured minimum buffer level. *Festive* employs a random scheduler and estimates bandwidth using a harmonic mean which is robust to outliers, this helps improve bandwidth estimation; additionally, the random scheduler requests segments independently of the player's start time, which improves the fairness between players operating in parallel. *MPC* solves an optimization problem for a number of segments ahead and uses throughput prediction. It requires both throughput estimation and buffer level to solve this, optimizing towards a set of defined QoE metrics. We have used the RobustMPC variant of MPC which is referred to as MPC in the remainder of this paper. *Bola* employs Lyapunov optimization techniques in order to minimize rebuffering and maximize video quality. It primarily uses buffer level to determine the appropriate bit rate for future segments.

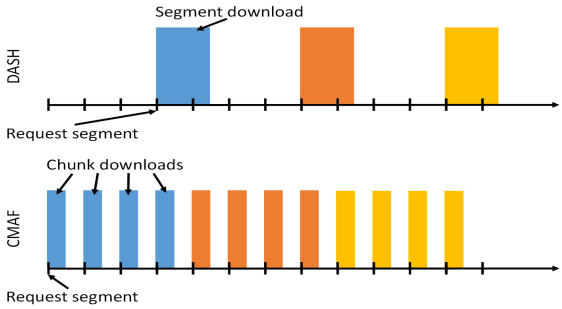

Figure 1: Timing diagram comparing segment delivery in DASH with and without CMAF.

2.4 QoE Factors

Quality of Experience factors in video streaming are an active area of research. Allan et al. [13] conducted a subjective test with 630 participants to find out how different QoE factors affect users. They concluded that rebuffering events are the most annoying to users, and that two short rebuffering events are more annoying than one long rebuffering event of the same duration. Garcia et al. [15] published a survey of QoE user studies, and also concluded that rebuffering events are the most annoying and that users would accept longer video start-up delay to get less rebuffering.

3 METHODOLOGY

We developed the simulation model described in this section to enable time-efficient evaluation of the performance of the four ABR algorithms, **Panda**[19], **Festive**[17], **MPC**[25], and **Bola**[24]. These were evaluated in DASH clients with and without the use of CMAF, with multiple settings of end to end latency and in many varied network conditions.

3.1 Simulation Framework

We developed the simulation model in NS-3 [8], extending an implementation of on-demand DASH [21] to: model live delivery of DASH and CMAF, use the four ABR algorithms, and to enable traffic shaping between the client and the server. The model, which is available on GitHub [9], has five configuration parameters, DASH/CMAF, ABR Algorithm, Throughput Trace, Live Delay, and Join Offset, which are used as described below.

The DASH/CMAF parameter determines whether the client acts as a regular DASH client or as a DASH client that supports the

CMAF format and chunked transfer encoding. When acting as a regular DASH client, the client requests segments from the server after they have been fully written to the server, they are then delivered by the server as whole segments. When acting as a CMAF client, the client requests segments from the server as soon as the first chunk of each segment has been written to the server; the server responds by delivering CMAF chunks as soon as they become available, emulating the delivery of CMAF chunks when transmitted using HTTP chunked transfer encoding.

The ABR Algorithm parameter determines which of the four ABR algorithms, Panda, Festive, MPC and Bola, is to be used by the client for future segment selection. The parameters of each ABR algorithm are set to the default parameters presented in their respective papers. The ABR algorithm simply provides the player with the quality at which to request the next segment.

The Throughput Trace specifies the file that contains timestamps and bandwidth values in kbps, which are used to adapt the bandwidth of the link between the client and server over time.

The Live Delay parameter specifies which segment the client requests first: a value of one indicates that the newest segment available on the server is requested first, two indicates the second newest available segment is the first to be requested, and so on.

The Join Offset parameter determines the time at which the client first requests a segment relative to the time at which segments are made available on the server. When set to 0s, the client requests a first segment as soon as a segment becomes available on the server, although which segment is requested is determined by the Live Delay parameter. When the segment duration is 2s and the Join Offset is set to 1s, the client requests a first segment mid-way between consecutive segments being made available on the server.

After the first segment has been delivered, the client requests subsequent segments as soon as the previous one has been delivered, or when they become available on the server, whichever is later.

The Live Delay and Join Offset parameters affect the end to end latency, that is, the time between a segment being encoded and the segment being played out by the client. Live Delay also affects the potential maximum buffer level of the client, as does Join Offset, but only with CMAF and a chunk duration less than Join Offset.

3.2 Video Encoding

We selected the first four minutes of the BigBuckBunny movie [1] and encoded it using x264 [11] at bitrates of {400, 800, 1200, 2400, 4800} (kbps) with resolutions {426x240, 640x360, 854x480, 1280x720, 1920x1080}. The encoded video was then segmented using MP4Box [7] into 2s segments for DASH, and into 2s segments with 0.5s chunks for CMAF-DASH. The resulting segments/chunks were used in DASH.JS framework described below, but the simulation model only required knowledge of their sizes.

3.3 Throughput Traces

We used CDN logs from the live BT Sport 1 service to set the bandwidth of the link between the client and server as a function of time. The CDN logs contained the request time, segment size and download time of each segment for every streaming session over a whole day. From these we produced a throughput trace file for each streaming session in the CDN logs, where each trace file contained pairs of timestamp, equal to the segment request time, and throughput, calculated as the size of the segment divided by its download time.

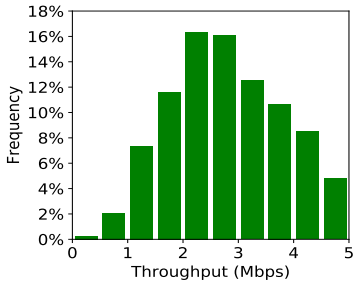

Figure 2: Histogram showing the distribution of throughput measurements below 5 Mbps in the 7,000 throughput traces.

We discarded trace files shorter than our four minute video clip, and cropped the others to the first four minutes. We discarded trace files that had mean throughput higher than our highest encoding bit rate of 4800 kbps. Additionally, since we wanted to study ABR performance, we discarded trace files where all of the throughput measurements were between the same two encoding bit rates.

This left 7,000 throughput traces to use in the experiments. The distribution of throughput in these traces is shown as a histogram in Figure 2, which shows the relative frequencies of throughput measurements below 5 Mbps. About 89% of the measurements are below the highest encoding bitrate of 4.8 Mbps.

3.4 Metrics

We used the following metrics to evaluate the performance of the four ABR algorithms. **Video Quality** is the arithmetic mean of the indices, in the range 0 to 4, of the quality at which the segments are requested. **Quality Variability** is the standard deviation of the encoded bitrates of the requested segments. **Rebuffer Ratio** is the ratio of the total rebuffering time to playback time. **QoE Score** is the overall score computed using the ITU-T Rec. P.1203 QoE model which combines bitrate, resolution, framerate and stall duration into a single value between 1 and 5 [22]. We have used a standalone implementation of the model found here [5].

4 VERIFICATION OF SIMULATION MODEL ACCURACY

To confirm the accuracy of our simulation model, we compared its performance with that of a real-time implementation based on the DASH.JS [4] player which we modified to allow us to test different delay settings. We enabled the use of the parameters Live Delay, which determines how far the player is behind the live edge, and Stable Buffer Time, which is the target buffer for the player; these are normally over-ridden when the player is in low-latency mode, which is required to deliver CMAF chunks using HTTP Chunked Transfer. The player was run in the Chromium browser [2].

We created a HTTP server in Node.js to serve pre-encoded DASH segments and CMAF chunks using HTTP/1.1 Chunked Transfer. Segments and chunks are served when they become available, as if the content were being encoded in real-time. Our server, which

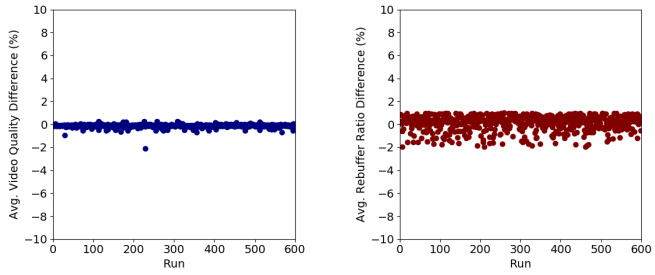

Figure 3: The average percentage difference in performance of the NS-3 and Dash.JS frameworks, measured using the QoE metrics, Video Quality (left), and Rebuffer Ratio (right).

is available on GitHub [3], was placed inside a virtual mininet [6] network, with the client located outside on a separate local network. These networks were connected by a link with bandwidth controlled using the linux tc module [10].

We compared the performance of the simulation model with that of the DASH.JS player using 100 throughput traces, with Live Delay set to 1 to 6 segments, and with the player configured to CMAF mode. In total, this resulted in 600 real-time runs with the four minute clip giving a total run time of around 42 hours.

Figure 3 shows the average percentage difference in performance of our simulation model and DASH.JS, when measured using the two QoE metrics: Video Quality, and Rebuffer Ratio. The left plot shows that the average Video Quality achieved by our simulation model and by DASH.JS on each of the 600 runs differ by less than +/-2%, with most differences being within +/-1%. The right plot shows that the average Rebuffer Ratio values achieved by our simulation model and by DASH.JS are also within +/-2% of each other, and most differences are between 0% and 1%.

Considering the difficulty of precise traffic shaping on a real network, we conclude that these results confirm that the simulation model is an accurate reflection of a real world system.

5 RESULTS

In this section we present a comparison of the performance of the four ABR algorithms, Panda, Festive, MPC and Bola, when used with DASH and CMAF, with Live Delay ranging from 1 to 4 segments, Join Offset equal to 0s, 0.5s, 1s, and 1.5s, and with the link between the server and the client being limited in turn by each of the 7,000 trace files. In total, we simulated the four minute clip being delivered to the client 896,000 times.

We report the performance of the four ABR algorithms using the QoE metrics previously described, averaged over the 7,000 runs with different throughput profiles. We combined Live Delay and Join Offset into a single value termed Join Delay, being equal to the time between the first segment being created and it being requested, measured in segment periods, and calculated as Live Delay added to Join Offset divided by two seconds.

5.1 Rebuffering

Figure 4 shows the percentage of sessions that experienced rebuffering as a function of Join Delay, for each of the four ABR algorithms

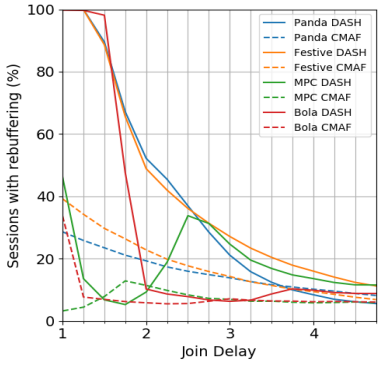

Figure 4: The percentage of sessions that experienced rebuffering as a function of Join Delay.

and for DASH and CMAF. The figure demonstrates that for most settings, all four ABR algorithms perform better with CMAF, with fewer sessions experiencing rebuffering. The best improvements occur at the lower values of Join Delay. For example, when Join Delay is 1 segment, the use of CMAF enables an additional 71%, 61%, 43%, and 66% of all sessions to avoid rebuffering, for Panda, Festive, MPC, and Bola respectively.

For all ABR algorithms except MPC, the percentage of CMAF sessions experiencing rebuffering generally decreases with increasing Join Delay. As Join Delay increases to 1.75, indicating a Join Offset of 1.5s, the percentage of all sessions experiencing rebuffering decreases by a further 7%, 13%, and 26% for Panda, Festive and Bola respectively. For MPC, the percentage of sessions with rebuffering increases as Join Delay increases from 1.0 to 1.75, and then decreases with increasing Join Delay. MPC outperforms Panda and Festive at all values of Join Delay.

Festive shows consistent but reducing benefit of CMAF over DASH as Join Delay increases, with 26% less sessions suffering rebuffering when Join Delay is 2.0 and only 7% when it is 4.0. Panda achieves decreasing benefit from CMAF over DASH as Join Delay increases, with 33% fewer sessions suffering rebuffering when Join Delay is 2.0, but with DASH slightly outperforming CMAF when Join Delay is 3.75 or more. MPC shows variable performance, especially with DASH, as initially the percentage of sessions suffering rebuffering decreases as Join Delay increases, but then rises to a peak at Join Delay of 2.5, before falling again. CMAF performance is better than DASH, except during the minimum Join Delay of 1.75. With Bola, CMAF performs much better than DASH when Join Delay is less than 2.0, but otherwise there is little difference.

Figure 6c shows the Average Rebuffer Ratio for all sessions as a function of Join Delay.

For Panda, Festive and Bola, the use of CMAF reduces the Average Rebuffer Ratio for low values of Join Delay. This benefit persists over the whole range for Festive, up to about 2.0 for Panda after which DASH suffers less rebuffering, and up to about 2.0 for Bola, after which DASH and CMAF alternate in terms of which causes more rebuffering.

MPC with either CMAF or DASH causes less rebuffering than the other three ABR algorithms across the range of Join Delay, with CMAF a little better for Join Delay greater than 2.0.

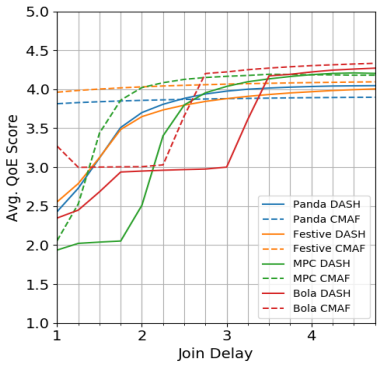

Figure 5: The average ITU-T Rec. P.1203 QoE Score of all sessions as a function of Join Delay.

5.2 Quality and Quality Variation

Figure 5 shows the average ITU-T Rec. P.1203 QoE Score of all sessions as a function of Join Delay, for each of the four ABR algorithms and for DASH and CMAF. The QoE Score generally increases with Join Delay for both DASH and CMAF. The QoE Score is higher when using CMAF for most settings with all ABR algorithms except Panda, where CMAF is better for Join Delays less than 2.5, and marginally worse otherwise. Generally there is more gain from using CMAF at low values of Join Offset, for example, improvements in QoE Score of 1.38, 1.4, 0.12 and 0.42 for Panda, Festive, MPC and Bola respectively when Join Delay is 1.0.

Panda and Festive with CMAF achieve a consistent QoE Score across the range of Join Delays, while with DASH, performance is significantly worse for Join Delays less than 2.0, illustrating the benefit of being able to fetch chunks earlier with CMAF. MPC and Bola also show this with their QoE Score, with DASH being shifted about 0.75 to the right of the respective CMAF versions.

Figure 6a shows the Average Quality Level of all sessions as a function of Join Delay. Panda and Festive achieved consistent an Average Quality Level across the range of Join Delay, with little difference between CMAF and DASH.

The curve for MPC with DASH is again approximately shifted to the right by 0.75 compared to MPC with CMAF, suggesting that MPC benefits from the earlier availability of chunks with CMAF. Bola exhibits similar shifted characteristics, but also shows different behaviour with very low values of Join Delay.

Figure 6b shows the Average Quality Variability of all sessions as a function of Join Delay. Again, Panda and Festive achieved consistent values across the range of Join Delays, but each showing less Average Quality Variability with CMAF than with DASH.

MPC again has a CMAF curve 0.75 to the left of the DASH curve. Average Quality Variability is low for Join Delays less than 2.0, because low quality is chosen consistently. But with both CMAF and DASH, the variation increases as Join Delay increases, as more buffering is available to allow quality to vary as a result of the MPC optimisation algorithm.

Bola also chooses low quality when Join Delay is low, and hence achieves low Average Quality Variability, but this increases beyond that of Panda and Festive at higher values of Join Delay.

6 DISCUSSION

ABR algorithm performance, with DASH and CMAF, generally improves as Join Delay increases, with higher video quality being requested and less rebuffering occurring; this is mainly due to the potential maximum buffer level increasing as Join Delay increases. For example, when Join Delay is one segment, the player is only one segment behind the live edge, restricting the player's maximum buffer level to one segment duration, giving only this time window for the player to detect and adjust to changes in network conditions.

Figure 7 shows the buffer level for two sessions with the same throughput trace, one for DASH and one for CMAF, where Join Delay is 2.0 and the Festive ABR algorithm is used. Buffer level was recorded after each segment had been fetched and again after each segment had finished playback. We observe that when using CMAF the average buffer level is higher, allowing it to avoid rebuffering at approximately 160s, unlike DASH, where multiple short rebuffering events occurred at this time. The higher average buffer level with CMAF is due to new segments being downloaded as chunks periodically every chunk duration, instead of being downloaded as segments periodically every segment duration. This also makes the buffer level more stable. With DASH, the buffer depletes by almost an entire segment duration before the next segment is fetched, after which the buffer level increases again by the segment duration. With CMAF the buffer depletes by almost an entire chunk duration before the next chunk is fetched, after which the buffer level increases again by the chunk duration.

Figure 7 also shows that after the period of rebuffering with DASH, the buffer level increases beyond the earlier peak because the end to end latency has increased by the total duration of rebuffering. This increases the potential maximum buffer level by the same amount, which the ABR algorithm takes advantage of from around time 200s.

In most cases, increasing Join Delay improves ABR performance with both DASH and CMAF. ABR performance is much higher with CMAF than with DASH when Join Delay is less than 2.0 and the player is close to the live edge. In this case, with DASH there are no additional segments available as they become available every segment duration, but with CMAF, where chunks become available every chunk duration, there are additional chunks available for download and playback. This can be seen in Figure 1.

Performance improvement with CMAF was not universal for all four ABR algorithms and settings. In general, the benefit from CMAF decreased as Join Delay increased, as the ABR algorithms performed well with DASH at high Join Delay. MPC and Bola benefited the most from CMAF, Festive achieved only a small improvement, and Panda was often better with DASH. This demonstrates the need for ABR algorithms to be designed taking CMAF into account to maximise the performance.

7 CONCLUSION

Live video streaming suffers higher latency than traditional broadcast, primarily due to the need for a large client buffer that allows the ABR algorithm to detect and respond to changes in network conditions in a timely manner. Improving ABR algorithm performance could allow a smaller buffer to be used, thereby reducing the latency for live streaming.

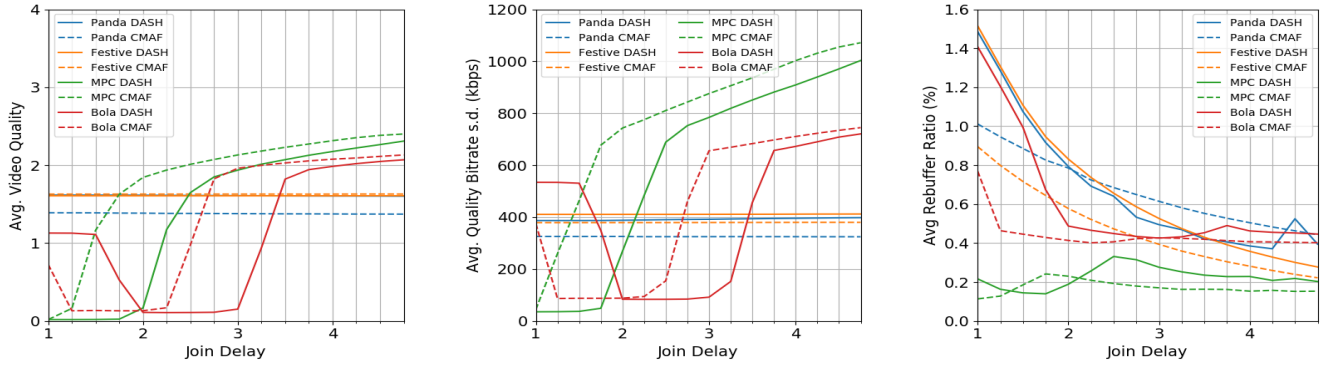

Figure 6: ABR performance as a function of Join Delay: a) Video Quality, b) Quality Variability, and c) Rebuffer Ratio.

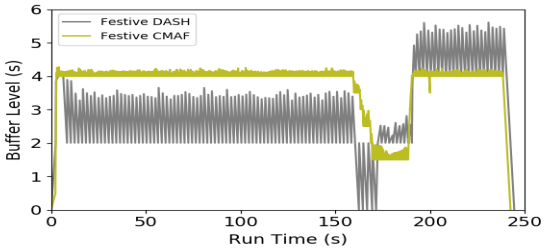

Figure 7: Buffer level of two Festive sessions using the same throughput trace, one using DASH and one with CMAF.

In this paper, we described a simulation model that supports live DASH and CMAF streaming and provided validation against a live implementation in DASH.js. We evaluated the impact of using CMAF rather than DASH on ABR performance for live video streaming, different end to end latency settings were used and server to client throughput was set according to data from the CDN logs of a commercial live television service provider. We conclude that the use of CMAF can significantly improve ABR performance at low latency, with a 43%-71% reduction in the number of sessions experiencing rebuffering with very low latency. However, we have also found that one of the four ABR algorithms tested, Panda, performed worse with CMAF across most settings, which highlights the need for ABR algorithms to be designed taking CMAF into account.

REFERENCES

[1] 2020. BigBuckBunny. https://peach.blender.org/.
[2] 2020. Chromium Web Browser. https://www.chromium.org/.
[3] 2020. CMAF Server. https://github.com/tomlyko/cmaf-server.
[4] 2020. DASH.JS. https://github.com/Dash-Industry-Forum/dash.js/wiki.
[5] 2020. ITU-T Rec. P.1203 Standalone Implementation. https://github.com/itu-p1203/itu-p1203.
[6] 2020. Mininet. http://www.mininet.org.
[7] 2020. MP4Box. https://gpac.wp.imt.fr/mp4box/.
[8] 2020. NS-3. https://www.nsnam.org/.
[9] 2020. A simulation model for Live DASH with CMAF chunks support. https://github.com/tomlyko/ns3-dash-cmaf-model.
[10] 2020. TC Module. http://manpages.ubuntu.com/manpages/xenial/man8/tc.8.html.
[11] 2020. x264. https://www.videolan.org/developers/x264.html.
[12] ISO/IEC 23009-1. 2019. Information technology-Dynamic adaptive streaming over HTTP (DASH)-Part 1: Media presentation description and segment formats. (2019).
[13] Brahim Allan, Mike Nilsson, and Ian Kegel. 2019. A Subjective Comparison of Broadcast and Unicast Transmission Impairments. *SMPTE Motion Imaging Journal* 128, 6 (2019), 1–15.
[14] 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, New York, NY, USA, 7–13. https://doi.org/10.1145/3304112.3325611
[15] M. . Garcia, F. De Simone, S. Tavakoli, N. Staelens, S. Egger, K. Brunnström, and A. Raake. 2014. Quality of experience and HTTP adaptive streaming: A review of subjective studies. In *2014 Sixth International Workshop on Quality of Multimedia Experience (QoMEX)*. 141–146. https://doi.org/10.1109/QoMEX.2014.6982310
[16] K Hughes and D Singer. 2017. Information technology–Multimedia application format (MPEG-A)–Part 19: Common media application format (CMAF) for segmented media. *ISO/IEC* (2017), 23000–19.
[17] J. Jiang, V. Sekar, and H. Zhang. 2014. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive. *IEEE/ACM Transactions on Networking* 22, 1 (Feb 2014), 326–340. https://doi.org/10.1109/TNET.2013.2291681
[18] J. Kua, G. Armitage, and P. Branch. 2017. A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP. *IEEE Communications Surveys Tutorials* 19, 3 (thirdquarter 2017), 1842–1866. https://doi.org/10.1109/COMST.2017.2685630
[19] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. 2014. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. *IEEE Journal on Selected Areas in Communications* 32, 4 (April 2014), 719–733. https://doi.org/10.1109/JSAC.2014.140405
[20] T. Lohmar, T. Einarsson, P. Fröjdh, F. Gabin, and M. Kampmann. 2011. Dynamic adaptive HTTP streaming of live content. In *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. 1–8. https://doi.org/10.1109/WoWMoM.2011.5986186
[21] Harald Ott, Konstantin Miller, and Adam Wolisz. 2017. Simulation Framework for HTTP-Based Adaptive Streaming Applications. In *Proceedings of the Workshop on Ns-3 (WNS3 '17)*. ACM, New York, NY, USA, 95–102. https://doi.org/10.1145/3067665.3067675
[22] Werner Robitza, Steve Göring, Alexander Raake, David Lindegren, Gunnar Heikkilä, Jörgen Gustafsson, Peter List, Bernhard Feiten, Ulf Wüstenhagen, Marie-Neige Garcia, Kazuhisa Yamagishi, and Simon Broom. 2018. HTTP Adaptive Streaming QoE Estimation with ITU-T Rec. P.1203 Open Databases and Software. In *9th ACM Multimedia Systems Conference*. Amsterdam. https://doi.org/10.1145/3204949.3208124
[23] 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
[24] 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
[25] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno 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