Llama - Low Latency Adaptive Media Algorithm

Tomasz Lyko, Matthew Broadbent, Nicholas Race
Lancaster University, United Kingdom
{t.lyko, m.broadbent, n.race}@lancaster.ac.uk

Mike Nilsson, Paul Farrow, Steve Appleby
British Telecommunications plc, United Kingdom
{mike.nilsson, paul.farrow, steve.appleby}@bt.com

***Abstract*—In the recent years, HTTP Adaptive Bit Rate (ABR) streaming including Dynamic Adaptive Streaming over HTTP (DASH) has become the most popular technology for video streaming over the Internet. The client device 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 high latency compared to traditional broadcast methods, mostly in the client buffer which needs to hold enough data to absorb any changes in network conditions. Clients employ an ABR algorithm which monitors network conditions and adjusts the quality at which segments are requested to maximise the user's Quality of Experience. The size of the client buffer depends on the ABR algorithm's capability to respond to changes in network conditions in a timely manner, hence, low latency live streaming requires an ABR algorithm that can perform well with a small client buffer. In this paper, we present Llama - a new ABR algorithm specifically designed to operate in such scenarios. Our new ABR algorithm employs the novel idea of using two independent throughput measurements made over different timescales. We have evaluated Llama by comparing it against four popular ABR algorithms in terms of multiple QoE metrics, across multiple client settings, and in various network scenarios based on CDN logs of a commercial live TV service. Llama outperforms other ABR algorithms, improving the P.1203 Mean Opinion Score (MOS) as well as reducing rebuffering by 33% when using DASH, and 68% with CMAF in the lowest latency scenario.**

I. Introduction

Video streaming has been the largest service on the internet for years and it continues to grow, this includes live video streaming, as seen in the most recent Cisco report [1].

At the moment, many video streaming services use HTTP Adaptive Streaming technologies such as MPEG Dynamic Adaptive Streaming over HTTP (DASH) [2], where content is split into short segments (usually from 2-10 seconds), encoded at multiple bit rates and then 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, which takes into account measurements of the network conditions to maximise the Quality of Experience (QoE) of the viewer. When using DASH for live content services, the client can only request segments after they become available on the server, as indicated by the manifest file.

The use of DASH for live video streaming usually causes high end to end latency compared to traditional broadcast methods due to client buffering to queue segments prior to playback to allow time for the ABR algorithm to adapt changes in network conditions. If the client buffer is depleted, the playback will stall, and the QoE will be reduced. For live streaming, the maximum amount of data that could be queued in the client buffer depends on the end to end latency, as only data that has been captured and encoded but not yet presented to the user can be buffered. For example, if the client is playing content three segment durations behind live, the maximum client buffer fill is three segments. Recently, the Common Media Application Format (CMAF) [3] has been standardised, enabling segments to be divided into chunks to aid low latency live streaming.

In this paper, we present Llama, an ABR algorithm specifically designed to operate at low buffer levels to offer better QoE for low latency live streaming scenarios. It can be used with HLS, DASH and CMAF. We compare the performance of Llama against four popular ABR algorithms in terms of multiple QoE metrics, across multiple client settings, and in various network scenarios based on CDN logs from a commercial live TV service. In order to perform such an extensive evaluation, we have developed and verified a simulation model which supports live DASH and CMAF.

II. Background and Related Work

A. ABR Algorithms

Bentaleb al. [4] published a survey of ABR algorithms used in HTTP Adaptive Streaming. They outlined the main goal of an ABR algorithm is to maximise viewer QoE. This involves trying to maximize the average video quality, while trying to minimize the number of rebuffering events, the time spent in the rebuffering state, and the frequency of changes of video quality. They note that most of these goals are in competition with each other, and therefore require a reasonable trade-off. This survey classifies client-side ABR algorithms into the following five classes: available bandwidth-based, playback buffer-based, proprietary solutions, mixed, and Markov Decision Process (MDP)-based. Available bandwidth-based ABRs select video quality based on estimates of the available bandwidth, usually by calculating the throughput of the previously fetched segment(s). Playback buffer-based ABRs select video quality based on the buffer level alone, usually only selecting a video quality once its buffer level threshold has been reached. Proprietary solutions include ABR algorithms from adaptive streaming solutions and player implementations, such as Apple HTTP Live Streaming (HLS) [5]. Mixed ABR algorithms include those that take into account multiple metrics such as bandwidth estimation, buffer level, and segment duration when selecting video quality. MDP-based ABR algorithms select video quality using the Markov Decision Process.

Popular ABR algorithms mentioned by the survey above are: **Panda** [6], **Festive** [7], **MPC** [8], and **Bola** [9]. *Panda* is an available bandwidth-based ABR which uses a probe-and-adapt approach similar to TCP's congestion control. It determines a target average data rate, based on which the appropriate video quality is selected. It monitors throughput and adjusts the target average data rate accordingly, as well as calculating the inter-request time for each segment to allow the buffer level to move towards the configured minimum buffer level. *Festive* is a mixed ABR which employs a random scheduler and estimates bandwidth using a harmonic mean. Harmonic mean is robust to outliers, which improves bandwidth estimation; additionally, the random scheduler requests segments independently of player's start time, which improves the fairness between players operating in parallel. *MPC* is a mixed ABR which 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, but will use the term MPC for the remainder of this paper. *Bola* is a playback buffer-based ABR that employs Lyapunov optimization techniques in order to minimize rebuffering and maximize video quality. It primarily uses buffer level to solve this optimization problem to calculate the appropriate video quality for future segments.

B. QoE Factors

Quality of Experience factors in video streaming are an active area of research. Allan et al. [10] conducted a subjective test with 630 participants to investigate how different QoE factors affect users. They concluded that rebuffering events were the most annoying to users. They also found that two short rebuffering events were more annoying to users than one longer rebuffering event of the same duration. Ghadiyaram et al. [11] found that rebuffering events at the beginning of the stream were less annoying to users than at the end, and attributed it to the hysteresis affect first observed by Seshadrinathan et al. [12].

Garcia et al. [13] published a survey of QoE user studies. They concluded that rebuffering events have the most negative impact on overall QoE. Frequent quality switches also decrease the overall QoE, as do abnormal quality changes; however, gradual quality changes can be acceptable to some users. The survey also concluded that users are willing to accept longer video start-up delay if it leads to less rebuffering.

Barman et al. [14] published the most recent survey on QoE factors in HTTP Adaptive Streaming. They concluded that rebuffering events are the most annoying to users, with both the duration and frequency of rebuffering events being relevant, and hence concluded that an ABR algorithm should aim to minimize both. They also noted some research suggesting that very short rebuffering events are not noticeable, and therefore, are less annoying to users. The survey mentions switching quality as another important QoE factor, where a high number of switches can have a negative impact on the overall QoE. They also noted that multi-level quality switches, where the quality jumps across more than one quality level, can also be detrimental to overall QoE.

C. Latency

Shuai et al. [15] found, that the main contributor to latency in adaptive streaming over HTTP is the client buffer. This buffer holds 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.

Lohmar et al. [16] outlined four sources of delay that are specific to HTTP streaming. The most significant is again 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 is the result of video being divided into segments which introduces a delay of at least one segment duration. CMAF reduces this source of delay to at least one chunk duration as segments can be further divided into smaller units [3]. Swaminathan et al. [17], as well as, Bouzakaria et al. [18] demonstrated how HTTP/1.1 Chunked Transfer, which allows for partial HTTP responses, can be utilized to reduce latency in regular DASH.

D. Common Media Application Format

The Common Media Application Format (CMAF) [3] allows a segment to be created as a sequence of chunks. Whereas a DASH segment must be completely written to a server before it is addressable and can be 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 bitrate at segment boundaries. 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.

Essaili et al. [19] and Viola et al. [20] demonstrated the reduction in latency due to CMAF chunks combined with HTTP/1.1 Chunked Transfer. In our previous work [21], we found that CMAF can improve ABR performance at low latency settings, enabling rebuffering to be reduced by 43%-71%. However, we also found that not all of the ABR algorithms we tested performed better with CMAF, and hence concluded that an ABR algorithm needs to be designed to taking CMAF into consideration.

TABLE I
ANALYSIS OF THROUGHPUT TRACES

	Total temp. increases	Total temp. decreases	Traces affected
Base measurements	7304	2572	3618
Moving arithmetic mean	108	16	113
Moving harmonic mean	37	6	43

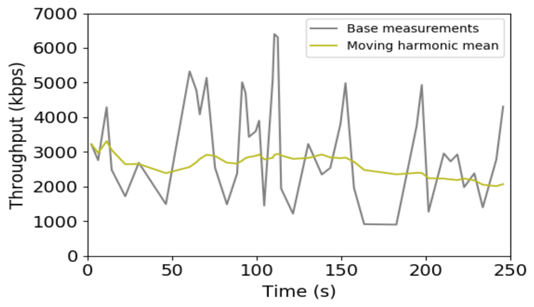

Fig. 1. Sample trace with base measurements and moving harmonic mean plotted.

As chunk delivery at the live edge is restricted by the encoder, estimation of network throughput is difficult for applications that have no direct visibility of any idle periods between chunks. Bentaleb et al. [22] attempted to solve this problem by ignoring throughput measurements for chunks that contain idle time in their download times. As our simulation model calculates the delivery time of chunks correctly, we do not consider this issue further.

III. Llama Design

Table I shows some characteristics of the 7,000 throughput traces we used in the experiment, and which are described in Section IV-C. The table shows the number of temporary increases in available bandwidth, where the bandwidth increases by more than 500 kbps and then decreases by more than 500 kbps within 10 seconds, as well as, the number of temporary decreases in bandwidth, where the bandwidth decreases by more than 500 kbps and then increases by more than 500 kbps in less than 10 seconds. The table also shows the number of throughput traces that experienced at least one temporary decrease or increase in bandwidth. The first row demonstrates how unstable the bandwidth is, with over half of the traces including at least one temporary change in bandwidth, and with a total of 7,304 temporary increases and 2,572 temporary decreases in available bandwidth. A smoothing function is often used to reduce the effect of temporary changes in bandwidth. The table shows the impact of smoothing using a moving arithmetic mean and a moving harmonic mean, each using 20 measurements over a period of 40s, showing that the moving harmonic mean is more effective, reducing the number of temporary changes in bandwidth and the number of traces that experience at least one temporary change. Figure 1 shows one of the traces with base measurements and moving harmonic mean plotted. It shows how using a smoothing function decreases the number of temporary changes in estimated bandwidth. At around 160s, the available bandwidth decreases for 20s, however, the moving harmonic mean smooths this out by estimating a higher value for the available bandwidth.

Our goal was to create an ABR algorithm designed specifically for low latency live streaming scenarios, where the client buffer level must be low to achieve the desired low latency. For example, when the client is operating only two segments behind live then its potential maximum buffer fullness is only two segments, which with segments of duration 2s, gives the ABR algorithm at most 4s to detect and respond to changing network conditions. In order for an ABR algorithm to perform well in such conditions, it needs to able to react quickly to adverse changes in network conditions, while perhaps being cautious when network conditions appear to be improving. Overall, the lower the latency the less time the ABR algorithm has to adapt to changing network conditions.

We have developed an ABR algorithm termed Llama which employs a novel idea of implementing two independent throughput measurements, one for decisions about reducing video quality and one for decisions about increasing video quality. This results in the ABR algorithm being quick to react to worsening network conditions in order to avoid rebuffering, but also being careful about increasing quality to improve video quality stability.

Our first design goal was to avoid rebuffering as it is the most detrimental factor to overall QoE. In low latency live streaming, to avoid rebuffering, the ABR algorithm needs to act on the first signs of deteriorating network conditions and reduce the quality at which segments are requested before the client buffer is depleted. As seen in Table I and Figure 1, using a smoothing function will eliminate some temporary decreases in throughput and hence prevent rapid response to worsening network conditions. Hence, Llama measures the bitrate at which the most recent segment was fetched and uses this to decide whether to switch to a lower quality representation, switching down to the next lower quality representation if the bitrate of the current representation is higher than the throughput of the last segment. This decision takes priority over all other ABR decisions.

Our second design goal was to increase quality stability as high quality variance can also be detrimental to overall QoE. As seen in Table I, if we had used the same throughput measurement as above to decide whether to switch to a higher quality, large variations in quality would occur as the quality would increase and decrease following temporary changes in available bandwidth. To avoid this issue we decided to implement a second throughput measurement calculated as the harmonic mean of the bitrate at which each of the past twenty segments had been fetched. Harmonic mean is robust to large outliers, but sensitive to smaller outliers and can provide a smoothed estimate of the available bandwidth. This estimate is used by Llama to decide whether to request the next segment at a higher quality, choosing to do so when the harmonic mean bitrate is higher than the encoded bitrate of the higher quality segment. This allows the ABR algorithm to avoid temporary increases in quality when the available bandwidth increases only for a short time.

Algorithm 1 Llama

```
1:  $harmonicMeanSize \leftarrow 20$
2:
3:  for $segment, n$ do
4:      $lastThroughput \leftarrow segmentThroughput(n-1)$
5:      $harmonicMean \leftarrow calculateHarmonicMean(n-1)$
6:      if $lastThroughput < currentQualityBitrate$ then
7:          Reduce quality representation by one if possible
8:      else if $harmonicMean > nextQualityBitrate$ then
9:          Increase quality representation by one if possible
10:     else
11:         Keep the same quality representation
12:     end if
13: end for
14:
15: function SEGMENTTHROUGHPUT($s$)
16:     $throughput \leftarrow segmentSize[s]/downloadTime[s]$
17:     return $throughput$
18: end function
19:
20: function CALCULATEHARMONICMEAN($s$)
21:     $sum \leftarrow 0$
22:     $sampleSize \leftarrow 0$
23:     for $i \leftarrow (s, s - harmonicMeanSize)$ do
24:         if $i > 0$ then
25:             $throughput \leftarrow segmentSize[i]/downloadTime[i]$
26:             $sum \leftarrow sum + (1/throughput)$
27:             $sampleSize \leftarrow sampleSize + 1$
28:         end if
29:     end for
30:     $mean \leftarrow sum/sampleSize$
31:     return $(1/mean)$
32: end function
```

Algorithm 1 demonstrates how Llama works. It has one parameter, *harmonicMeanSize*, which specifies the number of segments that are used for the harmonic mean calculation. The more segments that are used, the more smoothing is applied to the throughput measurements, and the more conservative are decisions to switch to higher quality. The use of fewer segments in this calculation would lead to higher mean video quality at the cost of more variation of quality. By default, the *harmonicMeanSize* is set to twenty segments, for both DASH and CMAF - where chunks are aggregated into segments. When there are fewer than *harmonicMeanSize* segments available, all available segments are used for the harmonic mean calculation. The quality representation is only changed if possible, that is, if there is a lower quality representation available when trying to decrease it, or if there is a higher quality representation available when trying to increase it. Otherwise, the quality representation stays the same.

IV. Methodology

In order to perform an extensive evaluation of all five ABR algorithms in reasonable time, we developed a simulation model which supports DASH and CMAF. The simulation model allows for faster than real-time evaluation as a single run of a four-minute video clip takes only 2-3 seconds in the simulation model compared to taking the whole duration of the video clip when using a real DASH player. In this section we describe our simulation model and its configuration parameters, as well as, the selected video encoding parameters and QoE metrics used for performance comparison.

A. Simulation Model

Our simulation model was developed in a discrete-event network simulator NS-3 [23] and is available on GitHub [24]. It supports live DASH, as well as, CMAF chunks. The implementation is based on an existing model [25] which supports on-demand DASH. We have extended the model to support live DASH, latency configuration, CMAF chunks, more ABR algorithms, and accurate traffic shaping between the client and the server. We have confirmed the accuracy of our simulation model by comparing it against a real DASH player in our previous work [21]. It has five parameters: Mode, ABR Algorithm, Throughput Profile, Live Delay, and Join Offset.

Mode sets the client into DASH or CMAF mode. When in DASH mode, the client requests segments from the server after they have been fully written to the server, and they are then delivered by the server as whole segments. When in CMAF mode, 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 five ABR algorithms, Llama, Panda, Festive, MPC and Bola, is to be used by the client for segment selection. Llama operates as described above, while the parameters of each of the other four ABR algorithms are set to the default values 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 most recent segment available on the server is requested first, two

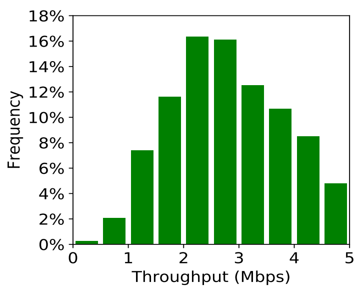

Fig. 2. Histogram showing the distribution of throughput measurements below 5 Mbps in the 7,000 throughput traces.

indicates the second most recent 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.

B. Video Encoding

We selected the first four minutes of the BigBuckBunny movie [26] and encoded it using x264 [27] at bitrates of {400, 800, 1200, 2400, 4800} (kbps) with resolutions {426x240, 640x360, 854x480, 1280x720, 1920x1080}. The encoded video was then segmented using MP4Box [28] into 2s segments for DASH, and into 2s segments with 0.5s chunks for CMAF. The resulting segments/chunks were used in DASH.JS framework described below, but the simulation model only required knowledge of their sizes.

C. Throughput Traces

We used CDN logs from a 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.

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 bitrate 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 bitrates.

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.

D. QoE Metrics

We used the following metrics to evaluate the performance of the five 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. **P.1203 MOS** is the overall Mean Opinion Score (MOS) computed using the ITU-T Recommendation P.1203 QoE model which combines bitrate, resolution, frame rate and stall duration into a single value between 1 and 5 [29]. We used a standalone implementation of the model [30].

V. Results

In this section, we present the results of our evaluation. We compare the performance of Llama against Panda, Festive, MPC and Bola firstly when the client is in DASH mode, and then when the client is in CMAF mode.

We have used the following parameters for both evaluations. The Live Delay was set to values ranging from 1 to 3 segments with the Join Offset set to 0s, 0.5s, 1s, and 1.5s for each Live Delay setting. All 7,000 throughput traces were used for each combination of Live Delay and Join Offset resulting in 84,000 runs per ABR algorithm for one mode of client. In total, the content was delivered to the client 840,000 times, including for all five ABR algorithms and both DASH and CMAF clients.

We report the performance of all 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.

A. DASH

For this evaluation, the client was configured in DASH mode and used with segments of duration of two seconds. Figure 3 shows the performance of all five ABR algorithms in terms of average Video Quality, Quality Variability, and Rebuffer Ratio, as well as, the percentage of sessions experiencing rebuffering for each value of Join Delay. Figure 4

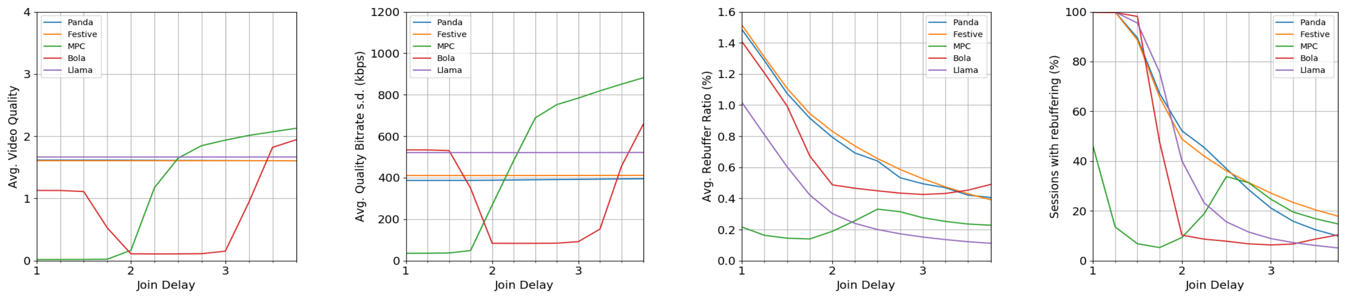

Fig. 3. The performance of the five ABR algorithms when used in DASH clients, in terms of average Video Quality, Quality Variability, and Rebuffer Ratio, as well as, the percentage of sessions experiencing rebuffering.

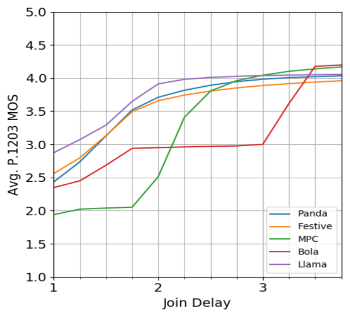

Fig. 4. Average P.1203 MOS as a function of Join Delay for each ABR algorithm when used in DASH clients.

shows the average P.1203 MOS as a function of Join Delay for each ABR algorithm.

Performance in terms of rebuffering. Llama achieved the second lowest average Rebuffer Ratio out of all five ABRs between Join Delays of 1 and 2, with MPC achieving the lowest. However, MPC also achieved the lowest average Video Quality for these settings, which we comment on later. Above Join Delay of 2 Llama outperformed all other ABRs. At Join Delay of 1 the average Rebuffer Ratio achieved by Panda, Festive, MPC, Bola, and Llama was 1.49%, 1.51%, 0.22%, 1.41%, and 1.02% respectively. At Join Delay of 2, average Rebuffer Ratio decreased to 0.79%, 0.83%, 0.19%, 0.49%, and 0.3% for Panda, Festive, MPC, Bola and Llama respectively. At Join Delay of 3, it decreased further for Panda, Festive, Bola and Llama to 0.49%, 0.53%, 0.43%, and 0.15% respectively. However, in case of MPC it increased to 0.28%.

Most ABRs had a high percentage of sessions with rebuffering events at the lowest value of Join Delay, and a decreasing percentage as the value of Join Delay increased. At Join Delay of 1, the percentage of sessions with rebuffering events for Panda, Festive, MPC, Bola, and Llama equalled to 100%, 100%, 40.16%, 99.74%, and 100% respectively. At Join Delay of 2, it decreased to 52.19%, 48.89%, 9.3%, 10.26%, and 40.13% for Panda, Festive, MPC, Bola, and Llama respectively. At Join Delay of 3, it decreased further for Panda, Festive, Bola and Llama to 21.16%, 27.14%, 6.33%, and 8.89% respectively, while for MPC the percentage of sessions with rebuffering events increased to 24.7%.

As Join Delay increased, the average Rebuffer Ratio and the percentage of sessions with rebuffering events for Llama always decreased, indicating that Llama is able to increase its performance by not only taking advantage of higher Live Delay settings, but also higher values of Join Offset.

Performance in terms of video quality. Llama achieved the highest average Video Quality for Join Delays between 1 and 2.5, and the second highest for Join Delays between 2.75 and 3.25 where MPC achieved the highest. Above Join Delay of 3.25, Llama achieved lower average Video Quality than MPC and Bola, the former one achieving the highest. At Join Delay of 1, the average Video Quality achieved by Panda, Festive, MPC, Bola, and Llama was equal to 1.62, 1.61, 0.02, 1.13, and 1.67 respectively. At higher values of Join Delay, Festive and Llama maintained the same average Video Quality, and Panda slightly decreased its average Video Quality to 1.61 at the highest Join Delay. MPC's average Video Quality stayed low for Join Delays up to 2 where it was equal to 0.17, and started to increase significantly at Join Delay of 2.25 where it increased to 1.18. MPC achieved the highest average Video Quality for Join Delays of 2.75 and higher, peaking at 2.13 when at the highest Join Delay. Bola's average Video Quality started to decrease at Join Delay of 1.5, and stayed low at less than 0.16 for Join Delays between 2 and 3. At Join Delay of 3.25 it started to increase again, peaking at 1.94 when at the highest Join Delay.

Llama achieved near constant average Quality Variability across all values of Join Delay, with differences within 0.51 and peaking at 521.7. Panda and Festive also achieved constant average Quality Variability, with differences within 7.88 and 0.65 along with peaks at 394.82 and 411.28 respectively. MPC's average Quality Variability was below 49.02 up to the Join Delay of 1.75, and significantly increased as Join Delay increased beyond 1.75 with peak value of 881.38 at the highest Join Delay. Bola had average Quality Variability of 534.08-530.62 for Join Delays of 1-1.5, which decreased between Join Delays of 1.5 and 2 to 83.9 and stayed within 0.6 up to Join Delay of 3. Its average Quality Variability increased beyond Join Delay of 3 and peaked at 656.61 when at the highest Join Delay.

Llama achieved the highest average Video Quality for values of Join Delay up to 2.5, and remained consistently high for

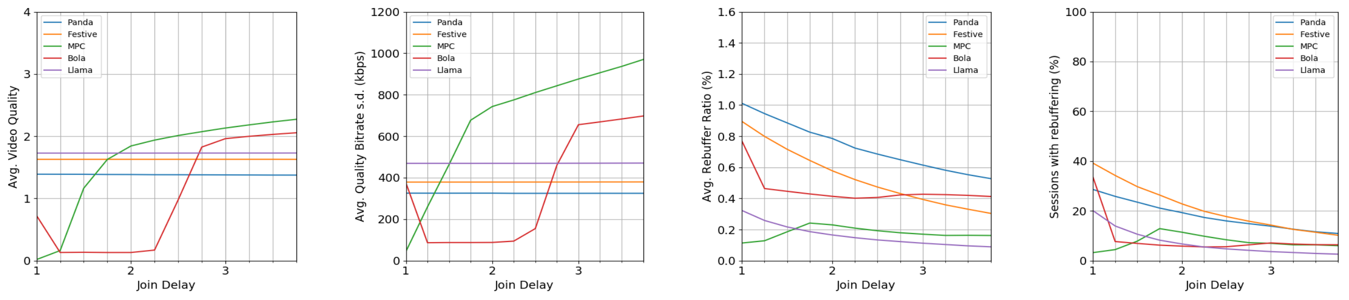

Fig. 5. The performance of the five ABR algorithms when used in CMAF clients, in terms of average Video Quality, Quality Variability, and Rebuffer Ratio, as well as, the percentage of sessions experiencing rebuffering.

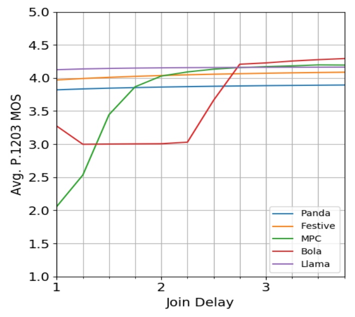

Fig. 6. Average P.1203 MOS as a function of Join Delay for each ABR algorithm when used in CMAF clients.

higher values of Join Delay. It also achieved near constant Quality Variability for all values of Join Delay resulting in much more stable Video Quality than MPC at Join Delay of 2.5 and higher.

Overall performance. For Join Delays between 1 and 2.75, Llama outperformed all other ABRs in terms of P.1203 MOS. At Join Delay of 3, it was still the highest, along with MPC which achieved the same P.1203 MOS. Beyond Join Delay of 3, Llama was outperformed by MPC and Bola which achieved slightly higher P.1203 MOS. At Join Delay of 1, Panda, Festive, MPC, Bola and Llama had P.1203 MOS of 2.43, 2.56, 1.94, 2.35, and 2.87 respectively. At Join Delay of 2, P.1203 MOS increased to 3.71, 3.66, 2.51, 2.95, and 3.91 for Panda, Festive, MPC, Bola, and Llama respectively. At Join Delay of 3 it further increased to 3.98, 3.89, 4.04, 3.0, and 4.04 with peaks of 4.03, 3.96, 4.17, 4.2, and 4.05 at the highest Join Delay for Panda, Festive, MPC, Bola, and Llama respectively.

B. CMAF

For this evaluation, the client was configured in CMAF mode and used with segments of duration two seconds and chunk duration of 0.5 seconds. Figure 5 shows the performance of all five ABRs in terms of average Video Quality, Quality Variability, and Rebuffer Ratio, as well as, the percentage of sessions experiencing rebuffering for each value of Join Delay. Figure 6 shows the average P.1203 MOS as a function of Join Delay for each ABR algorithm.

Performance in terms of rebuffering. For Join Delays between 1 and 1.5, Llama achieved the second lowest average Rebuffer Ratio and MPC achieved the lowest. For higher values of Join Delay, Llama achieved the lowest average Rebuffer Ratio. At Join Delay of 1, the average Rebuffer Ratio for Panda, Festive, MPC, Bola, and Llama equalled to 1.01%, 0.89%, 0.11%, 0.77%, and 0.32% respectively. At Join Delay of 2 it changed to 0.79%, 0.58%, 0.23%, 0.41%, and 0.17% for Panda, Festive, MPC, Bola, and Llama respectively. At Join Delay of 3 it changed further to 0.61%, 0.39%, 0.17%, 0.43%, and 0.11% for Panda, Festive, MPC, Bola, and Llama respectively.

Llama had less sessions with rebuffering events than Panda and Festive at all Join Delays, and the lowest percentage of sessions with rebuffering events at Join Delays of 2.25-3.75. At Join Delay of 1, the percentage of sessions with rebuffering events for Panda, Festive, MPC, Bola, and Llama was 28.61%, 39.2%, 3.24%, 33.64%, and 20.13% respectively. At Join Delay of 2, it increased to 11.44% for MPC, and decreased to 19.31%, 22.83%, 5.86%, and 6.73% for Panda, Festive, Bola, and Llama respectively. At Join Delay of 3, it increased to 7.14% for Bola, and decreased to 13.87%, 14.31%, 6.97%, and 3.66% for Panda, Festive, MPC, and Llama respectively.

With Llama, both average Rebuffer Ratio and the percentage of sessions with rebuffering decreased as the Join Delay increased, indicating that Llama is able to take advantage of higher Live Delay and Join Offset values to increase its performance with CMAF, just as it did with DASH.

Performance in terms of video quality. Llama achieved the highest average Video Quality at Join Delays between 1 and 1.75, and remained consistently high for higher values of Join Delay. It outperformed Panda and Festive at all values of Join Delay, but was outperformed by MPC for Join Delays of 2-3.75, as well as, by Bola for Join Delays of 2.75-3.75. At Join Delay of 1, the average Video Quality was equal to 1.39, 1.63, 0.02, 0.73, and 1.73 for Panda, Festive, MPC, Bola, and Llama respectively. It remained constant for Festive and Llama across all Join Delays, and near constant for Panda as it decreased by only 0.01 at the highest Join Delay. MPC's average Video Quality was low for the first two Join Delays, at 0.02-0.16, but started to significantly increase at the third Join Delay, outperforming all other ABRs at Join Delays of 2-3.75. Bola's

TABLE II
PERCENTAGE OF SESSIONS FOR WHICH LLAMA ACHIEVED BETTER P.1203 MOS THAN OTHER ABR ALGORITHMS

DASH				
Join Delay	Panda	Festive	MPC	Bola
1	90%	85%	98%	88%
2	57%	63%	98%	97%
3	42%	55%	40%	95%
CMAF				
Join Delay	Panda	Festive	MPC	Bola
1	92%	65%	99%	84%
2	92%	60%	66%	96%
3	90%	55%	37%	23%

average Video Quality at the lowest Join Delay was 0.73, then decreased to less than 0.18 for Join Delays between 1.25 and 2.25, and started to significantly increase at Join Delays of 2.5-3.75 - peaking at 2.06 when at the highest Join Delay.

Average Quality Variability achieved by Llama remained near constant across all values of Join Delay with differences within 1.24. This was also the case for Panda and Festive, with differences within 0.74 and 0.79 respectively. MPC's average Quality Variability was low at the lowest Join Delay, and sharply increased as the Join Delay increased, peaking at 969.65 when at the highest Join Delay - approximately twice as high as Llama's average Quality Variability. Bola's average Quality Variability was lower than Llama's up to Join Delay of 2.75, and higher for Join Delays of 3-3.75 - peaking at 697.36 when at highest Join Delay. Llama was much more stable than MPC for most values of Join Delay, and Bola for Join Delays values of 3-3.75. However, it was also slightly less stable than Panda and Festive.

Overall performance. Llama achieved the highest average P.1203 MOS for Join Delays of 1-2.5, and only up to 0.04 lower average P.1203 MOS than MPC for Join Delays of 2.75-3.75, where both of the ABRs were slightly outperformed by Bola. At Join Delay of 1, average P.1203 MOS achieved by Panda, Festive, MPC, Bola, and Llama was 3.82, 3.97, 2.06, 3.27 and 4.12 respectively. At Join Delay of 2, it decreased to 3.01 for Bola, and increased to 3.86, 4.04, 4.03, and 4.15 for Panda, Festive, Bola, and Llama respectively. At Join Delay of 3, it increased to 3.88, 4.07, 4.17, 4.23, and 4.16 for Panda, Festive, MPC, Bola, and Llama respectively.

VI. DISCUSSION

Llama outperformed the other ABR algorithms in terms of P.1203 MOS for both DASH and CMAF at the two lowest settings of Live Delay, which are crucial to get low latency.

With DASH, Llama achieved 0.31-0.93 better average P.1203 MOS than the other ABR algorithms when the Join Delay was equal to 1, that is, when the latency is just one segment duration, as well as achieving up to 33% less rebuffering. The only ABR algorithm which had lower average Rebuffer Ratio at these settings was MPC which achieved a value 78% lower than Llama. However, this low average Rebuffer Ratio is a result of MPC having significantly lower average Video Quality, equal to 0.02 and indicative of mostly choosing the lowest level of quality, compared to Llama's average Video Quality of 1.67. This is reflected in the average P.1203 MOS, with Llama achieving 0.93 higher than MPC. At Join Delay of 2, Llama achieved 0.2-1.4 better P.1203 MOS than other ABR algorithms, and up to 64% less rebuffering. Table II shows the percentage of sessions for which Llama achieved better P.1203 MOS than other ABRs at different Join Delay settings. Llama achieved better P.1203 MOS than the other ABR algorithms for 85-98%, and 57-98% of sessions at Join Delays of 1 and 2 respectively.

CMAF has been developed to enable reduced latency in live streaming, where the use of CMAF chunks can reduce the minimum latency from one segment duration to a single chunk duration. CMAF chunks combined with HTTP/1.1 Chunked Transfer Encoding allow for the use of small chunks without additional network overhead. We have tested our new ABR algorithm when used in a client which supports CMAF chunks. At Join Delay of 1 and 2, Llama achieved 0.15-2.06 and 0.11-1.14 better average P.1203 MOS than the other ABR algorithms, while also reducing rebuffering by up to 68% and 79% respectively. Table II shows Llama achieved better P.1203 MOS than the other ABR algorithms for 65-99%, and 60-96% of sessions at Join Delays of 1 and 2 respectively. These results clearly demonstrate that Llama can be used with CMAF and is capable of utilizing CMAF chunks to boost its performance.

One of the design goals for our new ABR algorithm was to improve video quality stability. This requires the ABR algorithm to minimise the number of unnecessary quality switches. In our results we can observe the impact of video quality stability on overall QoE. At the highest value of Join Delay, MPC achieved the highest average Video Quality and the second lowest average Rebuffer Ratio, but it did not achieve the highest average P.1203 MOS as this was negatively impacted by its frequent quality switches - which can be seen on the average Quality Variability charts where MPC was the highest by a significant margin. This was the case for both DASH and CMAF. In CMAF mode, at the highest Join Delay, Llama achieved 52% lower average Quality Variability and 24% lower average Video Quality than MPC, but its P.1203 MOS was only 0.03 lower.

Llama was outperformed by Bola and MPC at higher values of Join Delay in terms of average P.1203 MOS and Video Quality in both DASH and CMAF modes. In DASH mode at the highest Join Delay, when compared to Llama, MPC achieved 28% higher average Video Quality and 0.12 better average P.1203 MOS, while Bola achieved 16% higher average Video Quality and 0.15 better average P.1203 MOS. However, the average Rebuffer Ratio achieved by MPC and Bola was 109% and 346% respectively higher when compared to Llama. As seen in Table II, at Join Delay of 3, Llama achieved better P.1203 MOS for 40% and 95% of sessions when compared to MPC and Bola respectively in DASH, and for 37% and 23% of sessions when compared to MPC and Bola respectively in CMAF. Our new ABR algorithm is too conservative in terms of selecting higher quality bitrates when compared to MPC and Bola at higher values of Join Delay. In order for Llama to be deployed in environments without very low latency, it needs to be adjusted to make it less conservative. This could be done for example by reducing the number of segments used for the harmonic mean throughput calculation.

VII. Conclusion

In this paper, we have presented Llama - a new ABR algorithm specifically designed to operate in low latency live streaming scenarios. In such scenarios, the client buffer level is low, and hence, the ABR reaction time is severely limited. Llama employs a novel idea of using two throughput measurements made on different timescales, one to influence the decisions of whether to decrease the video quality and one for the decisions of whether to increase the video quality. This approach enables our ABR algorithm to react to deteriorating network conditions quickly, while maintaining stable video quality by ignoring temporary increases in available bandwidth. We have evaluated Llama by comparing it against four popular ABR algorithms, Panda, Festive, MPC, and Bola, with multiple client settings and using network scenarios based on 7,000 throughput traces obtained from CDN logs of a commercial live TV service. The performance of the ABR algorithms was presented using a range of QoE metrics, including the ITU-T Recommendation P.1203 QoE model which combines multiple QoE factors into a single value. At the two lowest Live Delay settings, giving latency of one and two segment durations, Llama outperformed the other ABR algorithms. In DASH mode, the P.1203 MOS improved by 0.31-0.93 and 0.2-1.4 respectively, and rebuffering reduced by up to 33% and 64% respectively, while in CMAF mode, Llama again outperformed the other ABR algorithms, achieving 0.15-2.06 and 0.11-1.14 respectively better P.1203 MOS and reducing rebuffering by up to 68% and 79% respectively.

References

[1] V. Cisco, “Cisco visual networking index: Forecast and trends, 2017–2022,” *White Paper*, vol. 1, 2018.

[2] I. 23009-1, “Information technology-dynamic adaptive streaming over http (dash)-part 1: Media presentation description and segment formats,” 2019.

[3] K. Hughes and D. Singer, “Information technology–multimedia application format (mpeg-a)–part 19: Common media application format (cmaf) for segmented media,” *ISO/IEC*, pp. 23 000–19, 2017.

[4] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann, “A survey on bitrate adaptation schemes for streaming media over http,” *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 562–585, 2019.

[5] R. 8216, “Http live streaming,” 2017. [Online]. Available: https://tools.ietf.org/html/rfc8216

[6] 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 Journal on Selected Areas in Communications*, vol. 32, no. 4, pp. 719–733, April 2014.

[7] J. Jiang, V. Sekar, and H. Zhang, “Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive,” *IEEE/ACM Transactions on Networking*, vol. 22, no. 1, pp. 326–340, Feb 2014.

[8] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, “A control-theoretic approach for dynamic adaptive video streaming over http,” *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 325–338, Aug. 2015. [Online]. Available: http://doi.acm.org/10.1145/2829988.2787486

[9] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, “Bola: Near-optimal bitrate adaptation for online videos,” in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[10] B. Allan, M. Nilsson, and I. Kegel, “A subjective comparison of broadcast and unicast transmission impairments,” *SMPTE Motion Imaging Journal*, vol. 128, no. 6, pp. 1–15, 2019.

[11] D. Ghadiyaram, J. Pan, and A. C. Bovik, “A subjective and objective study of stalling events in mobile streaming videos,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 1, pp. 183–197, 2019.

[12] K. Seshadrinathan and A. Bovik, “Temporal hysteresis model of time varying subjective video quality,” 06 2011, pp. 1153 – 1156.

[13] M. . Garcia, F. D. Simone, S. Tavakoli, N. Staelens, S. Egger, K. Brunnström, and A. Raake, “Quality of experience and http adaptive streaming: A review of subjective studies,” in *2014 Sixth International Workshop on Quality of Multimedia Experience (QoMEX)*, Sept 2014, pp. 141–146.

[14] N. Barman and M. G. Martini, “Qoe modeling for http adaptive video streaming–a survey and open challenges,” *IEEE Access*, vol. 7, pp. 30 831–30 859, 2019.

[15] Y. Shuai and T. Herfet, “Towards reduced latency in adaptive live streaming,” in *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan 2018, pp. 1–4.

[16] T. Lohmar, T. Einarsson, P. Fröjdh, F. Gabin, and M. Kampmann, “Dynamic adaptive http streaming of live content,” in *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, June 2011, pp. 1–8.

[17] V. Swaminathan and S. Wei, “Low latency live video streaming using http chunked encoding,” in *2011 IEEE 13th International Workshop on Multimedia Signal Processing*, 2011, pp. 1–6.

[18] N. Bouzakaria, C. Concolato, and J. Le Feuvre, “Overhead and performance of low latency live streaming using mpeg-dash,” in *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*, 2014, pp. 92–97.

[19] A. E. Essaili, T. Lohmar, and M. Ibrahim, “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)*, 2018, pp. 1–5.

[20] R. Viola, A. Gabilondo, A. Martin, J. F. Mogollón, M. Zorrilla, and J. Montalbán, “Qoe-based enhancements of chunked cmaf over low latency video streams,” in *2019 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, 2019, pp. 1–6.

[21] T. Lyko, M. Broadbent, N. Race, M. Nilsson, P. Farrow, and S. Appleby, “Evaluation of cmaf in live streaming scenarios,” in *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 21–26. [Online]. Available: https://doi.org/10.1145/3386290.3396932

[22] 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*, ser. NOSSDAV ’19. New York, NY, USA: ACM, 2019, pp. 7–13. [Online]. Available: http://doi.acm.org/10.1145/3304112.3325611

[23] “Ns-3,” 2020, https://www.nsnam.org/.

[24] “A simulation model for live dash with cmaf chunks support,” 2020, https://github.com/tomlyko/ns3-dash-cmaf-model.

[25] H. Ott, K. Miller, and A. Wolisz, “Simulation framework for http-based adaptive streaming applications,” in *Proceedings of the Workshop on Ns-3*, ser. WNS3 ’17. New York, NY, USA: ACM, 2017, pp. 95–102. [Online]. Available: http://doi.acm.org/10.1145/3067665.3067675

[26] “Bigbuckbunny,” 2020, https://peach.blender.org/.

[27] “x264,” 2020, https://www.videolan.org/developers/x264.html.

[28] “Mp4box,” 2020, https://gpac.wp.imt.fr/mp4box/.

[29] 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, “HTTP Adaptive Streaming QoE Estimation with ITU-T Rec. P.1203 – Open Databases and Software,” in *9th ACM Multimedia Systems Conference*, Amsterdam, 2018.

[30] “Itu-t rec. p.1203 standalone implementation,” 2020, https://github.com/itu-p1203/itu-p1203.