Multi-Source HTTP Live Streaming (MSHLS), an ABR Algorithm for Hybrid V2V-CDN Video Streaming

Ishani Sarkar, Guillaume Urvoy-Keller, Dino Lopez Pacheco, Soufiane Roubia, Quentin Jacquemart

Multi-Source HTTP Live Streaming (MSHLS), an ABR Algorithm for Hybrid V2V-CDN Video Streaming

Ishani Sarkar,
Quentin Jacquemart
and Soufiane Rouibia
Easybroadcast
Nantes, France

Guillaume Urvoy-Keller
and Dino Martin Lopez-Pacheco
Université Côte d'Azur
Laboratoire I3S CNRS
Sophia-Antipolis, France

***Abstract*—Live video streaming is gaining momentum and now represents a significant share of Internet traffic. Typical distribution architectures for this type of service rely on CDNs to meet the stringent QoS requirements of live video applications. The video is broken into small chunks, encoded at different quality levels, that the viewer downloads from a CDN server. As CDN-based solutions are costly to operate, hybrid architectures have been proposed, where video viewers are exchanging video chunks between each other (V2V mode), reverting to the CDN server only if the chunk could not be downloaded in V2V mode.**

Our focus in this work is on the design of the adaptive bit-rate (ABR) algorithm that governs the quality level requested by the viewer. We demonstrate that the ABR algorithm is facing specific challenges, and thus needs to be adapted for the hybrid V2V-CDN case. We introduce MSHLS, a new ABR algorithm that takes into account the difference of performance when downloading from another viewer or a CDN server in a live streaming context where the video buffers are kept small as clients need to remain synchronized when watching the stream.

We evaluate MSHLS using controlled and live experiments in a commercial hybrid V2V-CDN system and demonstrate that it outperforms the HLS public reference implementation, designed with a single source of content in mind.

I. Introduction

The adaptive bit-rate algorithms (ABR) are key tools that content providers use to optimise video quality. These algorithms run on the client side video player and dynamically choose the bit-rate for each video chunk. The different protocols (like the HTTP Live Streaming protocol –HLS– and the Dynamic Adaptive Streaming over HTTP protocol –DASH–) which use the ABR algorithm generally cut the video stream into chunks of (almost) fixed size.

When joining a stream, a client downloads a manifest file, which contains all the information about the different bitrates[1] in which the stream is encoded and the URLs to get these streams. Generally for live stream, the manifest files are updated periodically to adverstise the information about the new chunks. The use of small chunks, typically a few seconds, in live streaming, allows to change the quality of the stream depending on the network conditions. There are various conditions and goals which have been taken into consideration while developing an ABR algorithm in various literature [8, 7, 9, 4] which need to be kept in mind while developing an ABR algorithm:

- Low Re-buffering. Rebuffering occurs when the player consumes all the chunks in the buffer and results in a frozen video playback.
- Low Fluctuations, i.e., changes in the playing quality.
- Small Start-Up Time, which is the time before playout actually starts. A typical strategy is to start at a low to intermediate video quality and then switch to higher qualities when possible.
- Low-Latency. This criterion is specific to live streaming which requires that the time between publishing a video chunk and users watching it be kept small. This means that the size of the buffer data for live streaming video players is small and does not leave room for much error in the choice of the video quality bitrate.

Our focus in this work is on hybrid live video streaming where the traditional CDN distribution model is complemented with a viewer-to-viewer approach, whereby viewers can exchange video chunks, thus minimizing the distribution cost for the content owner. The hybrid V2V-CDN architecture allows users to receive chunks from two different sources, a CDN server or viewers connected to the same stream. Details on the hybrid V2V-CDN protocols used in this work, e.g WebRTC, are available in [6]. They key problem that we address in the present work is the design of an ABR algorithm suited for the need of hybrid V2V-CDN architectures.

While the ABR-based approaches discussed in the literature consider downloading from a single source [8, 7, 4], the same does not hold for hybrid viewer-to-viewer (V2V) protocols, where a client downloads either from another client watching the same video at the same quality or from a CDN server. The downloading throughput varies significantly between these two different sources, which poses unique challenges. As an example, we observed in [5], when profiling over 34,000 users,

[1]Using the legacy convention, we use birate when referring to the video encoding rate and throughput for the network transfer performance.

an average CDN throughput of around 10Mbps which drops to 2 Mbps in V2V mode. This results in huge variations in the network throughput observed by the ABR algorithm when switching from downloading from a CDN server to a mere viewer.

When devising an ABR algorithm suited for an hybrid V2V architecture, we must thus reconcile two conflicting goals: (1) allowing room for higher fluctuation in the network throughput when switching from one type of source to another, in order to increase the V2V efficiency, defined as the fraction of chunks downloaded in V2V mode and (2) quickly reacting to changes in the network throughput of the client to prevent depletion of the playout buffer. More precisely, we face the following challenges:

- The network throughput is going to be highly varying, due to the discrepancy among viewers and CDN servers network conditions. Thus we need some metrics which can dampen those high variations.
- At the same time, sudden decreases in the network throughput should be dealt very minutely. At the hybrid V2V-CDN protocol side, we already manage a $Timeout$ parameter when fetching a video chunk from another peer. This $Timeout$ allows us to have sufficient time to download from another peer but also keeps in check that if the video chunk cannot be received from this peer, we still have sufficient time to fetch it from the CDN.
- Since we consider live streaming, the buffer is adjusted to just store only 30 seconds of video, which is quite small as compared to VoD streaming where the buffer length is at least 90 seconds long. Furthermore, we also use the upload capacity of the user to upload in V2V mode to other viewers. Both of these parameters allow room for almost no error while deciding the bitrate for each chunk. Thus it is important to monitor closely the fluctuations in the network throughput and at the same time allow more room for the fluctuations caused by slow remote viewers.

The remaining of this paper is organized as follows. In Section II, we review the scientific and technical state of the art of the domain, before presenting in Section III our own algorithm, termed MSHLS for Multi-Source HTTP Live Streaming, that we evaluate through controlled and live experiments in Section V.

II. State of the Art

The default ABR algorithm against which we primarily compare MSHLS in section V is Apple's HTTP Live Streaming (HLS). HLS [1] exploits both the buffer occupancy and the download bandwidth estimation to select the appropriate video bitrate. The algorithm calculates the estimated bandwidth using an exponentially weighted moving average (EWMA) technique. The weight for historical data is 0.7 and the weight for present data is 0.3, which, we observed, can lead to high fluctuations in the hybrid case due to the discrepancy between CDN and and other viewers' throughput. Also, the HLS algorithms only calculates the EWMA when there are more than 2 chunks in the buffer ; otherwise it always try to download the smallest quality. This is a issue in the live streaming case in V2V mode, where the buffer is often relatively small. In our case, the buffer is 30 seconds, which corresponds to 5 chunks for typical channels where the chunk is 6-second long.

A number of advanced ABR algorithms have been proposed in the literature, e.g. [8, 4, 7]. They can be classified as buffer-based, throughput based and hybrid depending on the signal (throughput, buffer occupancy or both) that it used to trigger the change of requested quality level by the viewer. BOLA is a buffer based algorithm which monitors the buffer length to choose the bitrate for the next chunk [8]. It shares some commonalities with BBA [3] and was shown to outperform major representatives of the different families of ABR algorithms in [8]. The basic BOLA algorithm performs well only when the buffer occupancy is high, and not in case of low latency live streaming. To address these shortcomings, the BOLA-E variant has been devised, which adds or removes virtual chunks in the buffer to force changes in the bitrate [7]. Another version of BOLA called DYNAMIC was also proposed in [7], which uses the throughput signal to improve the performance of BOLA during the start-up phase.

There is a whole section of algorithms relying on machine learning models to predict the bitrate for the next chunks. CS2P [9] uses a machine learning to identify cluster of similar sessions in terms of throughout. The clustering step is done offline. For each cluster the prediction engine calculates the initial throughput by calculating the median throughput of the cluster. To improve the bitrate prediction in the middle of the stream, CS2P relies on a hidden Markov model. The initial throughput and the model can be combined at the video players and the ABR algorithm to predict the throughput of the users. They tested their algorithm on a video player based on the DashJS public implementation, both with actual players and using trace driven simulations.

Pensieve [4] relies on reinforcement learning to predict the bitrate of the next chunk. It takes as input the current throughput, buffer occupancy and chunk length and uses a QoE function as reward. The training of the algorithm is done using a simulation environment, which models the dynamics of video streaming. The Pensieve algorithm is then tested on both real world network traces and simulated network traces.

The authors in [10] also use reinforcement learning to select the bitrate. Since one of the major problem with machine learning is to find sufficient data to train the model, they created a publicly accessible website called Puffer, that live-streams six over-the-air commercial television channels. They assign each session to a different ABR algorithm, e.g. Pensieve [4]. With this approach, they were able to train the model with 38.6 years or data in just a few months. Although machine learning algorithms have shown great improvements, such algorithms are designed to operate at the server side as they are too computationally intensive for the players (esp. mobile devices). The authors of [4] propose an ABR server to offload the computation from the client to a central server, which however raises scalability issues.

III. Proposed Algorithm

MSHLS combines, similarly to previous client side algorithm, both the buffer and throughput signals to calculate the bitrate of the next chunk.

Rather than using directly the current buffer occupancy, we capture its minimum buffer value observed at the arrival times of the last N chunks in the buffer. The rationale behind this approach is to assess the minimum buffer length that needs to be maintained before switching to the next quality. It allows us to find the stable minimum buffer length in which the player works without going into re-buffering. The value of N is adjusted based on different conditions, and is discussed in more details in Section III-A.

We further use thresholds based on the buffer occupancy. Specifically, we divide the buffer occupancy into 3 regions: (i) Bin 1: less than 10% of max buffer length ; (ii) Bin 2: from 11% till 30% of max buffer length and (iii) Bin 3: more than 30% of the max buffer length. Remember that the buffer length is set to 30 seconds in our live-streaming case. We can see a visual representation of bins in % and seconds in Figure 1. In intuitive terms, bin 1 corresponds to a dangerous situation where buffer starvation can occur and we must take a critical decision. If in Bin 2, we maintain the current bitrate. When in Bin3, we take a decision to switch to a higher bitrate in case if some conditions are met. We detail the operations in those 3 bins below.

Bin 1 If the minimum buffer value falls in bin 1, i.e. less than 10% of the maximum buffer length (3 seconds), we shift to the lowest bitrate available to limit the chances of re-buffering. We choose the first bin value to be 10% because this state can lead to re-buffering with even a slight fluctuation in the network.

Bin 2 If the average minimum buffer value lies in the second bin i.e. less than 30% of the maximum buffer length (9 seconds), we remain in the current playing bitrate. This will avoid oscillations in the buffer quality that affect client experience if we switch back and forth continuously and, in the worst case, might lead to re-buffering.

Bin 3 The next bin is the largest one i.e., greater than 30% of maximum buffer occupancy (10-30 seconds). This bin represents the buffer values that are high enough for attempting an increase in the quality of the stream without starvation. It is in this bin that we handle the variations introduced by downloading from two different sources. When the buffer value lands in this bin, we further calculate the standard deviation (Std) of the N minimum buffer sample values and the average throughput over the same period of time. The standard deviation aims at capturing the fluctuations introduced by downloading from different sources:

- We use a function that finds the level which allows streaming using 80%[2] of the current bitrate.

[2]Similarly to the commercial ABR algorithms (HLS or DASH).

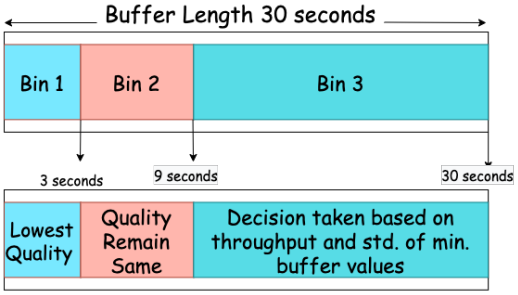

Fig. 1. Bin breakdown of buffer in MSHLS

- After identifying the quality level in which the current bitrate allows downloading the data, we check the standard deviation.
- If the standard deviation Std is less than an threshold SD (initially equal to 1.5), we select the quality previously chosen by the algorithm. If the standard deviation is greater than SD, we switch to the previous quality if the estimated bitrate does not allow the streaming in the current playing quality; otherwise we stay in the current playing quality. This ensures that there are no unnecessary oscillations due to the fluctuations that arise while downloading from different sources. Discussion on the value of the threshold SD is done in Section V-A3.

A. Adjusting the number of samples N

Setting the value N for the number of buffer level samples is challenging. On the one hand, a large value enables to dampen transient variations. On the other hand, we need to reduce it for the start-up of the session. We adopt the following strategies:

1) Start-up phase. The first three chunks are downloaded directly from the CDN. The quality chosen to download these three chunks is the so-called *auto quality* mentioned in the manifest file. For the channel used in the experiments for the present work, the *auto quality* is the medium quality. As we would like the ABR algorithm to quickly react, we set $N = 3$ at start-up. We next use $N = 5$.
2) Stationary phase. One of the drawbacks of the above approach is that for a period corresponding to the download of N chunks, the ABR algorithm just waits and collects data. In particular, it does not react to some sudden decrease in the network throughput. To address this issue, we take into consideration another characteristics of our V2V algorithm. The timeout for gathering a chunk in V2V mode is close to the duration of a chunk. For the channel used in our experiments, the chunk duration is 6 seconds and the timeout value is set to 5 seconds, which allows us time to revert to the CDN and download chunk in case of unsuccessful V2V download. The duration of the chunks remains almost the same for all quality levels.

To monitor that there is no sudden change in the network throughput, we monitor the buffer every T second (T is the duration of the last download), to check if some fresh data has reached the buffer. If after T seconds, there is no data, we immediately cancel the current download, check the current buffer length according to the bins, and if the buffer occupancy corresponds to the first two bins, we switch to the smallest quality to ensure that the buffer receives data before it is depleted. If it corresponds to bins 3 we switch to the previous quality level.

3) Rebuffering events. We probe the player every second to detect any rebuffering event and if this occurs, we immediately stop the current download and switch to the lowest quality.

IV. Experimental Strategy

A. Experimental Approach

The baseline scenario against which we compare MSHLS is the HLS public reference implementation. We use HLS because it is one of the most used streaming protocol in our production environment.

We performed experiments in a controlled environment, where clients run over physical servers in a Grid'5000 data center [2]. We used 4 nodes (physical machines) and used KVM along with cgroups to create isolated viewers. Each node hosts 15 viewers, for a total of 60 viewers. Each session lasts 30 minutes and we repeated each experiment 10 times.

The 60 viewers are watching a forked version of an existing channel dedicated to them that features three quality levels corresponding to 4, 7.2 and 10 Mbps respectively. For the considered stream, the HLS protocol uses a chunk duration of 6 seconds that we also adopted for our solution.

The controlled environment allows us to tune the access link characteristics of each client and assess how our algorithm performs for cases where all clients have similar or dissimilar access links. We use the traffic Control (tc) module of Linux to tune the access link characteristics.

It is not possible to perform reproducible experiments in the wild, e.g to adjust the ABR algorithm, HLS or MSHLS or parameters like the SD threshold. We however take advantage of our library that reports per client information to observe how our algorithm operates in the wild.

B. Upper bound on V2V efficiency

A key performance indicator for us is the V2V efficiency defined as the fraction of chunks downloaded in V2V mode.

We can compute an upper bound on the V2V efficiency taking into account the two following operational constraints: a viewer maintains connections with at most 10 other viewers (forming a swarm) and can upload to a maximum of 3 viewers simultaneously. These values of 3 and 10 have been observed, in our production system, to offer a good trade-off between the diversity of video chunks and the efforts needed to maintain those channels active.

We further assume that (i) the network performance of each viewer while uploading or downloading do not vary over time and (ii) a peer has an upload capacity equal to 3 times the video bitrate, i.e can upload continuously to three other viewers. In such a situation, the maximum achievable V2V efficiency is 70% and is obtained with the following scenario: three viewers download in a continuous manner the chunks from the CDN server and serve the 7 other viewers in the swarm.

V. Results

A. Experiments in a Controlled Environment

1) Impact of Access Network: In this scenario, we consider an ideal case where all clients are homogeneous in terms of access link characteristics. Specifically, we build three scenarios:

- The first experiment is done without restricting the upload or download capacity to allow the algorithm to choose the highest possible quality.
- For the second experiment, the download capacity is set to 8.5 Mbps and the upload capacity is 3×8.5 Mbps (as a client can send to 3 peers in parallel in the best case). This experiment is designed to force the algorithm to choose between the highest and middle quality.
- For the third experiment, the download capacity for the viewer is 4.5Mbps whereas the upload capacity is 3×4.5 Mbps. This experiment is designed to force the algorithm to choose between the middle and the smallest quality.

Table I reports the measured V2V efficiency as well as the number of video quality fluctuations (changes in quality level) for the default HLS implementation and MSHLS.

From Table I, we conclude that when there is no restriction on the bandwidth of the viewers, HLS fluctuates a lot, once every minute on average during the 30 minutes long experiments. This can be directly related to the discrepancy between the CDN and V2V throughputs. The results of MSHLS are in sharp contrast as there is 4.2 fluctuations on average over the entire experiment time. We further report in Table II the number of chunks downloaded in each quality level for HLS and MSHLS. We observe that MSHLS enables the clients to watch the highest quality for most of the playing time for the first (unconstrained) scenario.

For the second scenario, the fluctuations for the two algorithms become similar. However, HLS forces half of viewers to receive the smallest quality as can be seen in Table II. This is not the case with MSHLS where for 97% of the session time, clients remain in the middle playing quality, in line with their network link characteristics.

For the third scenario, where the bandwidth is between smallest and middle quality, all clients are in the lowest quality for both the algorithms.

From the above set of experiments we conclude that MSHLS allows the viewers to watch the content in the quality in line with its download capacity. MSHLS does not under-estimate or over-estimate the downloading bitrate and thus the playing quality. MSHLS performs much better than HLS algorithm wrt. the quality perceived by the viewers in

different scenarios and the V2V efficiency of the viewers is also improved with MSHLS. Since there are less fluctuations with respect to changes in quality of stream, the QoE for the viewer's is also improved.

	MSHLS		HLS	
	V2V %	Fluctuation	V2V %	Fluctuation
Scenario 1	65 +/- 2.1	4.2	43.27 +/- 4.1	34.2
Scenario 2	54.29 +/- 3.7	5.3	40.25 +/- 4.28	6.4
Scenario 3	45.2 +/- 2.9	0	38.35 +/- 4.56	0

TABLE I
V2V%, STANDARD DEVIATION OF V2V AND AVERAGE FLUCTUATION

	MSHLS			HLS		
	Highest	Medium	Smallest	Highest	Medium	Smallest
Scenario 1	467.11	14.6	21.2	176.23	110.32	212.469
Scenario 2	28.2	453.20	20.14	96.12	182.48	229.32
Scenario 3	0	0	510	0	0	514

TABLE II
NUMBER OF CHUNKS PER QUALITY LEVEL

2) Competition among clients (with different access links): Experiments in the previous section was focused on homogeneous scenarios where all clients have the same access link. The next set of experiments consists in mixing clients able to download at the highest quality level (termed good clients) and intermediate quality level (termed bad clients).

The objective is to put MSHLS under pressure as the throughput and buffer occupancy are going to fluctuate more over time depending on whether a client is receiving a chunk from a good client, a bad client or the CDN server. We do not present HLS results in this case as its performance significantly degrades, leading to good clients stuck in the middle or low quality.

One could a priori think that good and bad clients will be independent from each other and won't interact together. This is however not the case as when the clients, good or bad, joined the session, they start with the *auto quality*, which is the middle quality in the case of this stream. Good clients will thus, upon arrival, interact with bad clients. When this happens, they receive the chunks very late which affects the buffer occupancy measured by MSHLS. In contrast, when the good clients receive the chunk from either CDN or good viewer, the minimum buffer length increases but this also increases the standard deviation estimated in MSHLS.

The objective here is to thus observe the interplay between good and bad clients and to check if good clients can indeed converge to their expected quality level. We consider three scenarios:

- 50% of the clients are good viewers i.e. their download capacity is not restricted and 50% of the viewers are bad viewers, i.e. their download capacity is restricted to just above the middle quality level (8.5 Mbps).
- 25% bad viewers and 75% good viewers.
- 75 bad viewers and 25% good viewers.

Obviously, the more bad clients surround the good clients, the more difficult it is for the algorithm to take the right decision. There is thus an increasing complexity as the fraction of bad clients increases. Figures 2 to 4 present typical trajectories for the three scenarios. For these figures, we have aggregated the playing quality of each viewer for each chunk over slices of 5 minutes. Since the player quality for each viewer has been aggregated for 5 minutes, it is possible that a single client is present in two or three different quality levels based on the fluctuations it observes for this slice.

For the first experiment, the V2V efficiency (over all experiments) for this case is 67.94%, close to the 70% upper bound, with an average number fluctuation over the 30 minutes of 7.72. Almost 95% of the good clients were able to achieve the highest quality. Results are similar for the second experiment: 66.37% of V2V efficiency and average fluctuation of 8.94. The last scenario is the more challenging and in about one third of the cases, the good clients are not served at the highest quality, see Figure 4. We also observe more fluctuations between the middle and smallest quality. Thus the viewers in the above heterogeneous experiments do not converge to the their ideal playing quality trajectory and the average fluctuations for the playing quality increased too.

This lead us to further investigate the impact of the SD threshold (initially equal to 1.5) that should be large enough to account for the multi source scenario, but not to large to react to actual network fluctuations.

3) Impact of the SD threshold: The SD parameter plays a key role to give a chance to the good clients to converge to a higher video quality. In the section, we focus on the most challenging heterogeneous scenario with 75% of bad clients and observe the impact of adjusting the SD parameter, by increasing the SD value to 3 and then to 5.

When $SD = 3$, around 70% of the good viewers achieved the highest quality as compared to just 30% with SD value of 1.5. Also, on average over 90% of the bad clients achieve the middle quality as compared to 66% when $SD = 1.5$. When increasing SD to 5, we observe from Figure 6 that almost 95% of the good clients reach the highest quality and every bad client is also able to reach the middle quality.

Based on the results obtained in the controlled environment, we decided to deploy MSHLS in the wild with an SD threshold of 3, as it significantly outperforms the initial 1.5 case. Remember that using a too high SD value might affect the reactivity of the algorithm to actual changes in the network conditions. To assess the behavior of the algorithm in realistic conditions, we report observations on specific or random clients operating the new version of the library in the next section.

B. Tracing of clients in the wild

We tested MSHLS on two different streams being used in production for different clients. Both the streams feature 3 different playing qualities that correspond to 0.56, 1.19 and 2.01Mbps for stream 1 and 0.7, 1.15, 1.89Mbps for stream 2. While these rates are smaller than the ones used in the

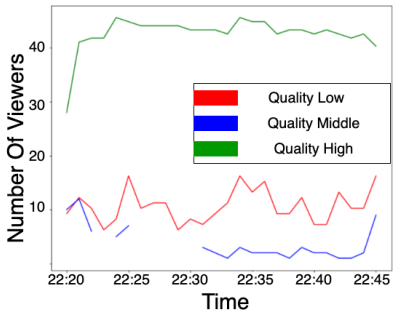

Fig. 2. 45 Good 15 Bad Viewers

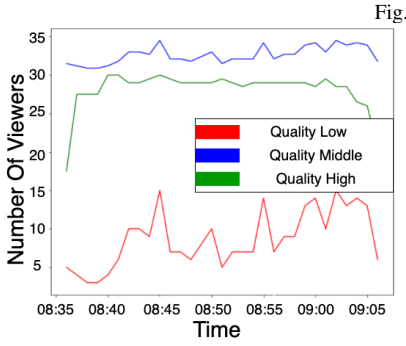

Fig. 3. 30 Good 30 Bad Viewers

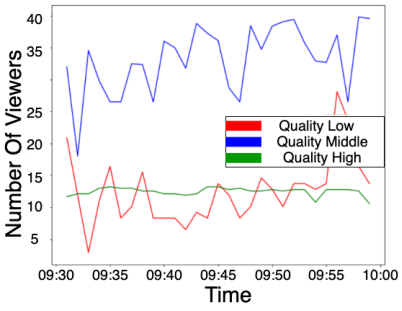

Fig. 4. 15 Good 45 Bad Viewers

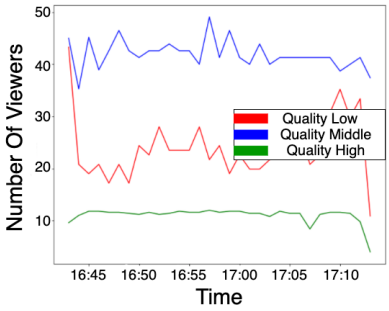

Fig. 5. Heterogeneous Clients, SD = 3

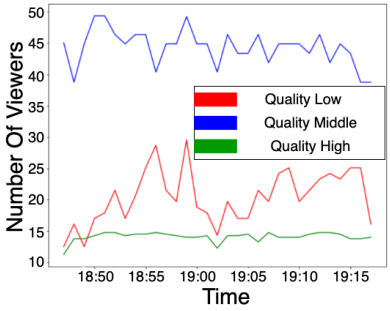

Fig. 6. Heterogeneous Clients, SD=5

controlled experiments, they are representative of the channels operated by Easybroadcast, that originate for the majority from Africa, while client are mostly in North Africa or Europe.

We watched the stream with a client under our control (called *local viewer*) and captured all the minimum buffer values and standard deviation values calculated by the algorithm to closely monitor the decisions taken by MSHLS. Our local machine has an FTTH connection to the Internet and thus operates in favorable network conditions.

We further compared the result of the local viewer with the ones of the other viewers in the same channels during the same time period, either with some aggregate statistics or by tracking the trajectories of random viewers. An advantage of the local client compare to the random clients is that we have access to more statistics, e.g., the exact time series of buffer samples, while for random clients, we only have the averages over 30 seconds time slices.

1) Throughput: We first characterize the throughput observed for the two channels. The throughput of the local viewer for the first stream is 5Mbps on average for V2V transaction whereas the CDN throughput was around 125Mbps. For the second stream, the values are 4.1Mbps and 83Mbps respectively. The time series of observed V2V and CDN throughputs are presented in Figure 7 and Figure 8 respectively. We can thus clearly see that the bitrate calculation of MSHLS will feature high variations.

As for the other viewers connected to the same streams during the same time period, we observed, for the first stream, an average V2V throughput of 3.83Mbps whereas the CDN throughput is 30.86Mbps. For the second stream, the values are 3.0Mbps and 13.93Mbps respectively. The time series of throughput values (one sample corresponds to 30 seconds) of all clients are presented in Figure 9 and Figure 10. While the local client with FTTH access does experience much higher network throughput as compared to the viewers in the wild in

general, most client still have enough throughput to download chunks from either CDN or V2V mode.

This overall picture indicates that the quality of experience perceived by the clients should be quite high for these two channels. This is confirmed by the overall fraction of chunks in each quality level. Concerning the first stream, 76% of the chunks were downloaded in the highest quality, 12% in the middle quality and 11% of the chunks in the smallest quality. For the second stream, 84% of the chunks were downloaded in the highest quality, 6% in the middle quality and 10% in the lowest quality. In the next section, we will look at individual trajectories of viewers to exemplify the behavior of MSHLS in the wild.

2) Buffer Length Variations and Playing Quality: The minimum buffer values are calculated by the algorithm over a period of $N = 5$ chunks. The length of each chunk being 6 seconds (with slight variations), this roughly corresponds to 30 seconds.

Let us first focus on the local viewer under our control. For the first stream, we report in Figure 11 the average and median of the minimum buffer values over 30-second period. In Figure 12 we see the standard deviation calculated and the decisions taken by the algorithm. Each time the deviation gets higher than the thresold ($SD = 3$), the algorithm changes the playing quality. For the second stream, we observe the same behavior, see Figures 13 and 14. While these changes of quality may be deemed as false positives, we observe that they occur relatively infrequently. We consider them as a price to pay to maintain the reactivity of MSHLS to more significant network changes. We investigated the changes of network conditions by observing the trajectory of random clients in the wild, as it is difficult emulate the actual changes faced by clients in a control environment as we did in Section V-A

Since, in the wild, the current version of the library does not capture the minimum buffer values or the standard deviation calculated by the algorithm, we cannot plot the same graphs as for the local viewer. We can however report the buffer value (over 30-second slices) and the playing quality of the stream of random clients.

We picked four clients, two for the first stream and two for the second stream that experienced different network conditions, leading to different dynamics in their buffer. Their trajectory in terms of buffer and playing quality (right-side of graphs, 3 is the highest quality, then 2 and 1) are presented in Figures 15 to 18. The duration of the graphs is a function of the session duration: approximately 3,5 hours for viewers 1 and 4 and 1 hour for viewers 2 and 3. We can make the following observations:

- Viewer 1 experiences the highest buffer occupancy. As such, its quality level is often the highest (3).
- Viewers 2 and 3 achieve lower performance than viewer 1, which result in buffer occupancy in the range $[10s, 15s]$, which corresponds to bins 2 and 3 (Figure 1). MSHLS still manages to keep them often in the highest quality and we don't often observe low values which could indicate buffer starvations.
- Viewer 4 has buffer occupancy often in bin 1 (less than 3 s) and bin 2. Consequently, MSHLS maintains it most of the time in the intermediate to lower quality.

In conclusion, in the cases exemplified by these 4 viewers, we observe that MSHLS manages to maintain high quality levels with low fluctuations as the rates of the four clients are respectively: 0.14 per mn for viewer 1, 0,375 for viewer 2, 0.2 for viewer 3 and 0.23 for viewer 4. Interestingly, the V2V efficiency remains acceptable for all clients. The overall efficiency for stream 1 is 40% and 30.7% for stream 2. As for the four viewers, we obtain: 41.36% for viewer 1, 31.57% for viewer 2, 25.83% for viewer 3 and 26.7% for viewer 4. While not a formal proof of the efficiency of MSHLS in the wild, these case studies however underscore that the algorithm offers a goof trade-off between quality level and V2V efficiency.

VI. Conclusion and Future Work

The HLS algorithm is not meant to handle the bandwidth fluctuation that arises from downloading from different sources of content, esp. if their throughput is very different. MSHLS does a better job in handling the discrepancy between the CDN and V2V downloads. It improves the V2V efficiency, which is the good news for the content owner, and at the same time improves the user experience.

MSHLS has two important parameters. First, the number of samples N that governs the time horizon over which we assess the buffer occupancy. The other key parameter is the SD threshold value that allows to account for the variations when downloading from a CDN server or another viewer. Overall, using these parameters, MSHLS offers a good trade-off between the reactivity to network fluctuations and the ability to receive from sources with different performance.

We demonstrated the effectiveness of MSHLS using controlled experiments with all the clients watching a stream under our control and also through the observation of clients in the wild. The latter enables to validate the behaviour of MSHLS when facing actual changes in the network conditions that are difficult to emulate in a controlled environment.

For future work, we intent to test our algorithm against all the DASH algorithms as well. We would also like to extend our algorithm for the VoD streams, still for an hybrid V2V-CDN scenario.

References

[1] https://github.com/video-dev/hls.js/blob/master/docs/api.md.

[2] Raphaël Bolze, Franck Cappello, Eddy Caron, Michel Daydé, Frédéric Desprez, Emmanuel Jeannot, Yvon Jégou, Stephane Lanteri, Julien Leduc, Noredine Melab, et al. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. *The International Journal of High Performance Computing Applications*, 20(4):481–494, 2006.

[3] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: evidence from a large video

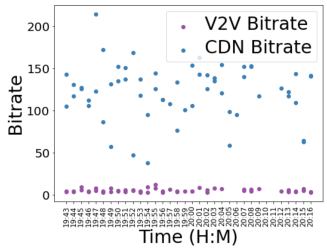

Fig. 7. Throughput Local Viewer: Stream1

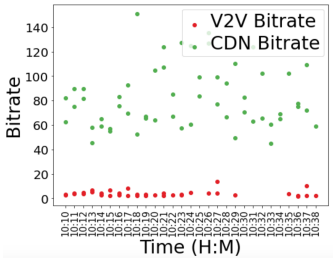

Fig. 8. Throughput Local Viewer: Stream2

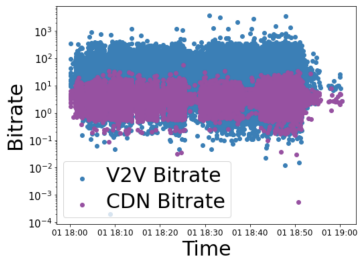

Fig. 9. Throughput Wild Viewers: Stream1

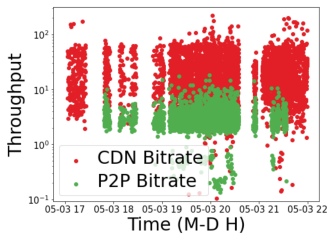

Fig. 10. Throughput Wild Viewers: Stream2

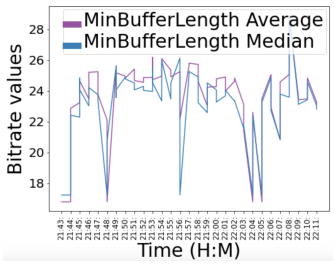

Fig. 11. Avg. Minimum Buffer Values(Local Viewer): Stream1

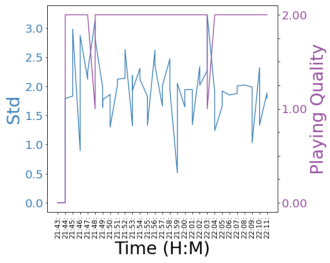

Fig. 12. Std and Playing Quality(Local Viewer): Stream1

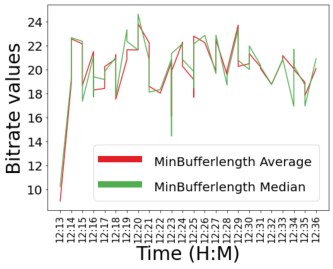

Fig. 13. Avg. Minimum Buffer Values(Local Viewer): Stream2

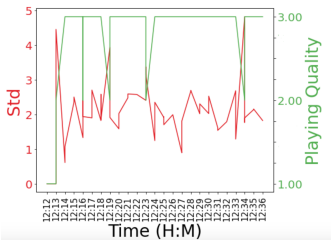

Fig. 14. Std and Playing Quality(Local Viewer): Stream2

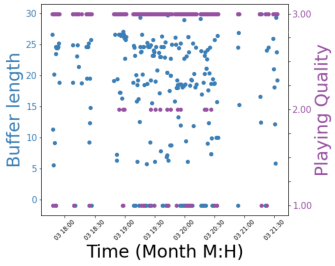

Fig. 15. BufferLength & PlayingQuality,Viewer1: Stream1

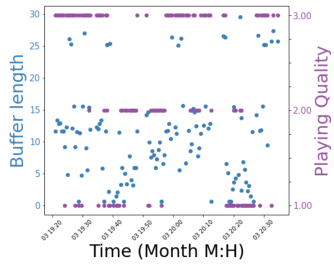

Fig. 16. BufferLength & PlayingQuality,Viewer2: Stream1

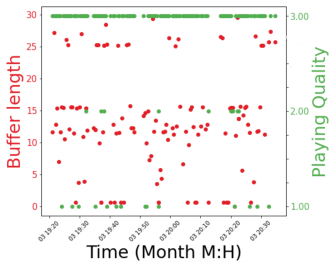

Fig. 17. BufferLength & PlayingQuality,Viewer3: Stream2

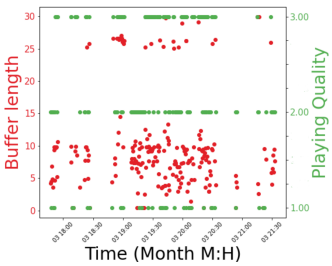

Fig. 18. BufferLength & PlayingQuality,Viewer4: Stream2

streaming service. In Fabián E. Bustamante, Y. Charlie Hu, Arvind Krishnamurthy, and Sylvia Ratnasamy, editors, *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*, pages 187–198. ACM, 2014.

[4] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*, pages 197–210. ACM, 2017.

[5] Ishani Sarkar, Soufiane Roubia, Dino Martin López-Pacheco, and Guillaume Urvoy-Keller. A data-driven analysis and tuning of a live hybrid CDN/V2V video distribution system. In Oliver Hohlfeld, Andra Lutu, and Dave Levin, editors, *Passive and Active Measurement - 22nd International Conference, PAM 2021, Virtual Event, March 29 - April 1, 2021, Proceedings*, volume 12671 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 2021.

[6] Ishani Sarkar, Soufiane Rouibia, Dino Lopez Pacheco, and Guillaume Urvoy-Keller. Proactive information dissemination in webrtc-based live video distribution. In *16th International Wireless Communications and Mobile Computing Conference, IWCMC 2020, Limassol, Cyprus, June 15-19, 2020*, pages 304–309. IEEE, 2020.

[7] Kevin Spiteri, Ramesh K. Sitaraman, and Daniel Sparacio. From theory to practice: improving bitrate adaptation in the DASH reference player. In Pablo César, Michael Zink, and Niall Murray, editors, *Proceedings of the 9th ACM Multimedia Systems Conference, MMSys 2018, Amsterdam, The Netherlands, June 12-15, 2018*, pages 123–137. ACM, 2018.

[8] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. BOLA: near-optimal bitrate adaptation for online videos. *IEEE/ACM Trans. Netw.*, 28(4):1698–1711, 2020.

[9] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. CS2P: improving video bitrate selection and adaptation with data-driven throughput prediction. In Marinho P. Bar-

cellos, Jon Crowcroft, Amin Vahdat, and Sachin Katti, editors, *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*, pages 272–285. ACM, 2016.

[10] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning in situ: a randomized experiment in video streaming. In Ranjita Bhagwan and George Porter, editors, *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, pages 495–511. USENIX Association, 2020.