Comparison of Adaptive Bitrate Algorithms for Multi-user DASH Video Streaming

Hai-Dang Nguyen

***Abstract*—Dynamic Adaptive Streaming over HTTP (DASH) has currently become a de facto standard for streaming video over the Internet. It contributes towards to leveraging the network friendliness of TCP protocol with firewall and bandwidth sharing. Further more, splitting video in multiple fix-duration segments and pre-encoding them in multiple quality levels introduce a flexible way for bitrate adaptation in various network conditions.**

Recently, a number of Adaptation BitRate (ABR) algorithms have been proposed; most of them, however, was simply evaluated in single-client cases. In practice, a network system may consist of multiple users, which compete for a bottleneck link. In such scenarios, clients may exhibit un-predicted behaviours, which can cause the fluctuation of perceived bandwidth even when network conditions remain unchanged. In this work, we indicate several biases among clients in multi-user scenarios. We also give a look inside state-of-the-art ABR algorithms and evaluate their performance in multi-user cases in terms of stability, efficiency and fairness. Finally, some improvements was introduced to achieve a better trade-off between aforementioned performance metrics.

***Index Terms*—Bitrate adaptation; HAS; DASH; adaptive video streaming; ABR schemes.**

I. Introduction

RECENTLY, HTTP streaming has become cost-effective method for multimedia delivery over the Internet [1]. Due to the bandwidth fluctuation nature of TCP protocol [2], which is the underlying layer of HTTP, one of the most important requirements for any streaming systems is bitrate adaptability. To serve for that purpose, a video is chunked into multiple segments, which uniformly last for a fixed duration. A video segment, in turn, is encoded in multiple representations (versions) that differ in bitrate or resolution. A metadata file is also generated to contain characteristics of representations. In a streaming session, A client first fetches the metadata file in order to get information about the video and its segments. An ABR algorithm, which is installed in client side, then decides which video representation should be requested at each time step based on current network conditions or terminal capabilities [3].

Numerous ABR algorithms have been proposed to be implemented in client side. They are clustered in three categories: bandwidth-based, buffer-based and mixed adaptation [4]. Bandwidth-based adaptation makes representation decisions based on its measured bandwidth, which can be calculated as the size of a fetched video segment divided by the its download time. This type of adaption is strongly influenced by bandwidth fluctuation. Meanwhile, buffer-based adaptive algorithms uses the client's buffer occupancy to select the appropriate version of the next segment. They can absorb short-term oscillations of bandwidth to achieve better bitrate stability. Finally, mixed adaptation makes bitrate selections based on the combination of network conditions and buffer occupancy.

In multi-user scenarios, the competition between clients and the discrete nature of bitrate representations lead to the difficulty in precisely measuring the fair-shared bandwidth [5]. Several methods have been proposed to address this problem [5] [6]; however, their bitrate selections are simply based on bandwidth-based adaptation. Meanwhile, buffer-based algorithms [19] [20], which outperform bandwidth-based ones in in terms of efficiency and stability, have not yet evaluated in multi-user scenarios. In this work, we make comparison between the aforementioned adaptive algorithms in a network of multiple clients. We also propose a new algorithm with some improvements in order to achieve better trade-off between performance metrics.

The paper is organized as follow. Section II gives some backgrounds on HTTP Adaptive Streaming. Section III provides related work to give brief review of adaptation methods. Our proposed improvements on state-of-the-art algorithms are detailed in Section IV. Comparisons between algorithms are discussed in Section V. Section VI concludes the paper.

II. Background

A. HTTP Adaptive Streaming System

The system architecture of a DASH system is shown in Figure 1. At server side, a video content is encoded in multiple versions with different birate levels. Each version then is split into temporal segments. In a streaming session, at first, the client sends a HTTP request to the server in order to obtain metadata file, which contains the information of all video representations (alternatives or versions). A rate adaptation logic at client side, where a ABR algorithm is installed, decides the next segment's bitrate based on several criteria such as measured bandwidth and buffer occupancy at each time step. The video segments are then delivered from the server to the client via a sequence of request-response transactions [1]. Once a video segment is decoded completely in the client, it will be stored in the client's buffer before being played out.

Several factors may impact on Quality of Experience (QoE) in HTTP Streaming. Average bitrate is one of influence factors [9]. Authors in [10] also indicate that QoE is strongly influenced by the frequency and duration of buffer depletion, which causes playout freezes. Besides, the sudden drop of

video bitrate causes adverse impacts on user's experience. [11].

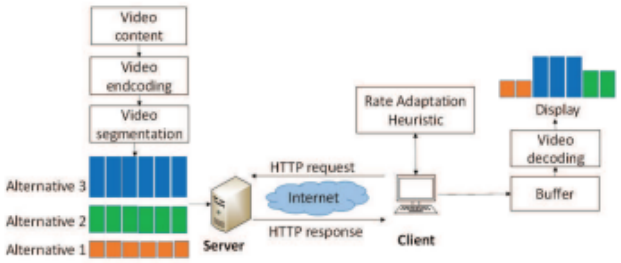


Fig. 1: DASH system architecture.

B. System Model

A video is chopped into temporal segments of fixed duration τ seconds. Each segment has been pre-encoded at L video bitrates, all stored at the server. Denote by $R := R_1, ..., R_L$ the set of available video bitrates, with $0 < R_l < R_m$ for $l < m$.

At the beginning of each download step n, a rate adaptation algorithm will:

- Selects the video bitrate of the next segment to be downloaded, $r[n] \in R$.
- Specifies how much time to give for the current download, until the next download request (i.e., the inter-request time), $\hat{T}[n]$.

Let $\tilde{T}[n]$ be the download duration. The next download step starts after the duration of $T[n]$, which is calculated as follow:

$$T[n] = \max(\hat{T}[n], \tilde{T}[n]) \tag{1}$$

That is if the download duration $\tilde{T}[n]$ shorter than target delay $\hat{T}[n]$, the client have to wait $\hat{T}[n] - \tilde{T}[n]$ until the next download. Figure 2 illustrates the downloading process in two scenarios. At the beginning of each downloading time step, the scheduled duration $\hat{T}[n]$ and selected bitrate $r[n]$ are updated. In Scenario A, when the segment is downloaded in a duration $\tilde{T}[n]$ shorter than the scheduled duration $\hat{T}[n]$, then the client have to wait until the next time step to take new decisions. Otherwise, in the Scenario B, if the download time is longer than expected, then the client will start downloading next segment immediately.

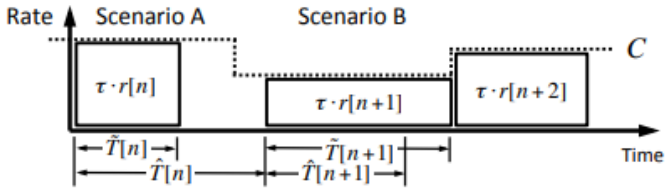


Fig. 2: Segment downloading process.

Once the client has already received the whole segment data, it can measure the network bandwidth $\hat{x}[n]$ as data size divided by download time:

$$\hat{x}[n] = \frac{r[n] \times \tau}{\tilde{T}[n]} \tag{2}$$

where $x[n]$ is selected bitrate at time step n.

The buffer occupancy in video streaming is calculated in second other than data size. Assume that the client plays out the video segments at nominal speed, i.e, if a video segment lasts for τ seconds, the client takes exactly τ seconds to completely play out it. At each downloading time step n, the client receives one more segment of τ seconds but also consumes $T[n]$ seconds of video compared to last time step $n-1$. Therefore, the buffer occupancy update is formulated as follow:

$$B[n] = \max(0, B[n-1] + \tau - T[n]) \tag{3}$$

The maximum operation here indicates that the buffer occupancy cannot be negative.

III. Related work

A. ABR in multi-user scenarios

Client-based adaptive bitrate algorithms can be divided into three categories based on the the signal they use to decide the next segment's bitrate: bandwidth-based, buffer-based and mixed adaptation [4]. Follow [5], a algorithm should consist of two parts:

- Measure the next segment bitrate.
- Schedule the request for the next video segment.

While a simple rate adaptation algorithm works fairly with a single client, multiple clients competing a bottleneck link could cause undesirable problems [6]. There are two sources of bias in multi-user case. Firstly, with periodic request intervals, the initial condition can cause the unfairness in bandwidth allocation. Secondly, a client requesting higher bitrate can estimate higher bandwidth. This is because there are points that it occupy the bottleneck link alone, and it can consume whole network resource [6]. The explanation of such biases is detailed below.

There are two popular options for scheduling: immediate download and periodic download [6]. For the first option, the client will download the next chunk immediately after the previous chunk has been downloaded completely. Greedily downloading the low bitrate chunks will ramp-up buffer quickly and cause the buffer overflow. Furthermore, in live streaming, the next chunk cannot be available immediately. In the second option, the chunks is downloaded periodically, so that the buffer is remained at optimal level. The issue with this approach is illustrated in the Figure 3. Suppose that the players A, B and C compete for a bottleneck link. Players B and C request periodically at even seconds while player A requests at odd seconds. Obviously, players B and C must compete to each other and observe only half of bandwidth while player A is able to use the whole bandwidth. In other words, the initial condition can cause the unfairness in bandwidth allocation.

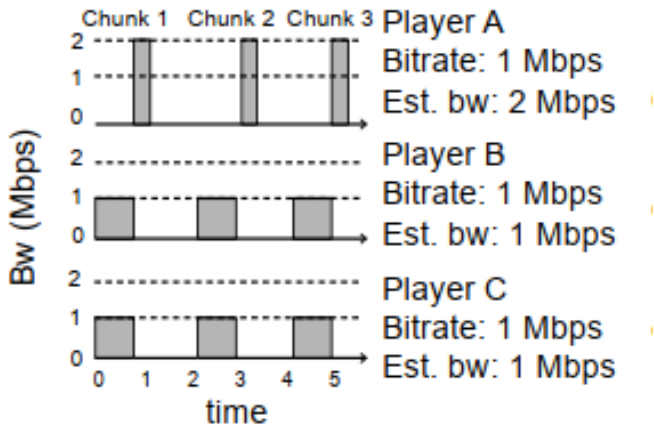


Fig. 3: Initial condition bias

The second type of bias is illustrated in Figure 4. In this case, player A requests higher bitrate compared to player B and C. It can be seen that there are time instants where player A uses the whole bandwidth while this is not the case for the others. Thus, player A will estimate bandwidth higher than players B and C. In other words, players currently opting higher bitrate will perceive higher bandwidth.

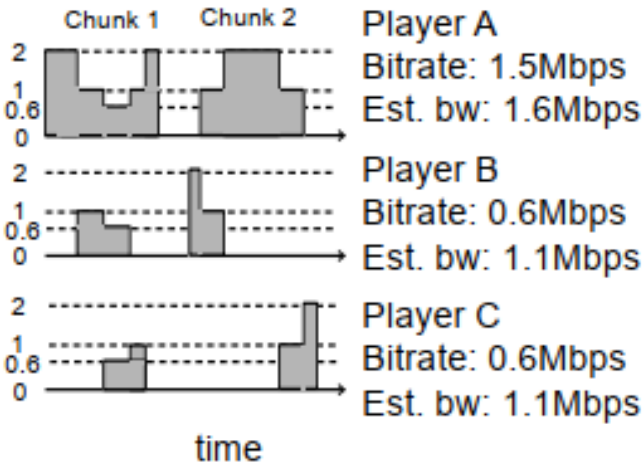


Fig. 4: Bitrate selection bias

Follow [5], in multi-user case, a robust adaptive algorithm must achieve three following goals:

- Fairness: Clients competing a bottleneck link will eventually converge to an equitable network resource.
- Efficiency: Clients must choose the highest bitrate as possible to maximize bandwidth usage.
- Stability: Clients should avoid the unnecessary bitrate switches as it causes annoyance for users.

The following subsections give a look inside state-of-the-art ABR algorithms. The explanation of notions used in algorithms is detailed in Table I.

TABLE I: Notions used in algorithms

Notions	Explanation
τ	Segment duration
$r[n]$	Selected bitrate for segment at time step n
$\hat{x}[n]$	Estimated bandwidth for segment at time step n
$\tilde{x}[n]$	Actual bandwidth of segment at time step n
R	Set of bitrate $R = \{R_1, R_2, ..., R_L\}$
$T[n]$	Actual inter-request time of segment at time step n
$\hat{T}[n]$	Scheduled inter-request time of video segment at time step n
$\tilde{T}[n]$	Download duration of video segment at time step n
$B[n]$	Buffer level after completely downloading of segment at time step n (in seconds)
$b[n] = \frac{B[n]}{\tau}$	Buffer level after completely downloading of segment at time step n (in number of segments)
B_{min}/B_{max}	Minimum/Maximum client buffer duration
w	Probing additive increase bitrate
k	Probing convergence rate
α	Smoothing convergence rate
β	Buffer convergence rate

B. Bandwidth-based adaptation

In this type of scheme, the client makes its decisions based on measured bandwidth.

1) Conventional methods: Almost of today's commercial players [13] [14] [15] implement the measuring and scheduling parts of the ABR algorithm in a similar way. Authors in [5] generalize conventional ABR into a 4-step algorithm as described in Algorithm 1 and use it as a benchmark.

Algorithm 1: Conventional Algorithm

At the beginning of downloading n-th chunk

1) Estimate the bandwidth share $\hat{x}[n]$ by equating it to measured bandwidth:

$$\hat{x}[n] = \tilde{x}[n-1] \tag{4}$$

2) Smooth out $\hat{x}[n]$ to produce filtered version $\hat{y}[n]$ by function S.
3) Quantize $\hat{y}[n]$ to the discrete video bitrate $r[n]$ by function Q.
4) Schedule the next download request via

$$\hat{T}[n] = \begin{cases} 0 & B[n-1] < B_{max} \\ \tau & otherwise \end{cases} \tag{5}$$

First, the algorithm estimate the fair-share bandwidth by equating it to the measured bandwidth, which is calculated as in Equation 2.

In order to avoid short-term fluctuation of bandwidth, the algorithm smooths out the estimated bandwidth $\hat{x}[n]$ to get the filtered version $\hat{y}[n]$ by smoothing function S.

In the next step, the continuous smoothed bandwidth $\hat{y}[n]$ is mapped to discrete video bitrate $r[n]$ using quantization function Q. In practice, function Q can use previous fetched bitrates and buffer occupancy as side information to make decisions.

Finally, the algorithm determines the target inter-request duration $\hat{T}[n]$. If the buffer occupancy $B[n-1]$ is smaller than the buffer size B_{max}, then $\hat{T}[n]$ is set to 0, which means that the next segment is downloaded right after the current segment is finished. Otherwise, $\hat{T}[n]$ is set to segment duration τ to keep the buffer level around buffer size B_{max}.

The main drawback of conventional algorithm is that in multi-user scenarios, the measured bandwidth does not always represent the fair-shared bandwidth due to the biases presented in subsection A. Furthermore, the periodic scheduling scheme cannot address the problem of initial condition bias.

2) FESTIVE: FESTIVE algorithm [6] follows the same steps as Conventional one. In the second step, it smooths out estimated bandwidth by taking harmonic mean over last 20 samples, which is formulated by Equation 6.

$$\frac{1}{\hat{x}[n]} = \frac{1}{20}\sum_{i=n-20}^{n-1}\frac{1}{\hat{x}[i]} \tag{6}$$

The harmonic mean is proved for its robustness against large outliers [16].

To mitigate the problem of Bitrate selection bias in subsection A, FESTIVE introduces the stateful bitrate selection, which is based on the idea that the player with lower rate will ramp up more aggressively. The video segment bitrate is updated as following conventions.

- The bitrate at level k will increase to level $k+1$ only after k segments. This guarantee that players with lower bitrate will ramp up faster than ones choosing higher bitrate.
- The bitrate at level k will decrease to level $k-1$ if current bitrate is larger than 0.85 $\times$ smoothed bandwidth $\hat{y}[n]$.

However, the bitrate selected as above is only used for reference. Switching between bitrate levels too frequently can cause annoyance for user [17]. Therefore, at each time step, the client decides to stick on the current bitrate r_{curr} or to switch to reference bitrate r_{ref} by computing the following trade-off. The efficiency cost for bitrate r is:

$$score_{efficiency}(r) = |\frac{r}{min(\hat{y}[n], r_{ref})} - 1| \tag{7}$$

The score is best and equal to 0 when $r = r_{ref}$. It means that we will fully utilize the bottleneck link if choosing the r_{ref}. The stability cost for a given bitrate r is a function of the number of bitrate switches the player has undergone over last 20 seconds, which is denoted by s.

$$score_{stability}(r) = \begin{cases} 2^s + 1 & if \quad r = r_{ref} \\ 2^s & if \quad r = r_{curr} \end{cases} \tag{8}$$

Obviously, algorithm will add more penalty of adding new switch if there have already many switches in the past. The player computes the combination of two scores for current bitrate and reference rate and pick the lower one.

$$score_{combined}(r) = score_{stability}(r) + \delta \times score_{stability}(r) \tag{9}$$

where δ is a tunable knob to control the trade-off between efficiency and stability.

In order to address problems caused by periodic scheduling, FESTIVE uses Randomized scheduling. In particular, instead of keeping a constant interval between two consecutive requests, it introduces a randomness in deciding start time of the next request. Specifically, after completely downloading segment $n-1$, a target buffer size $rbuf_{n-1}$ is randomly chosen following uniform distribution from the range $(B_{max} - \tau, B_{max} + \tau]$. Let t_{n-1}^{start} be the time when the n-th segment is requested, t_{n-1}^{end} be the time that it is downloaded. Then the time for next chunk request is scheduled as follow.

$$t_n^{start} = \begin{cases} t_{n-1}^{end} & B[n-1] < rbuf_{n-1} \\ t_{n-1}^{end} + B[n-1] - rbuf_{n-1} & otherwise \end{cases} \tag{10}$$

3) PANDA: PANDA algorithm [5] also follow steps in Algorithm 1. Two main contributions of PANDA are in estimating and scheduling steps. PANDA algorithm is detailed in Algorithm 2.

Algorithm 2: PANDA Algorithm

At the beginning of downloading n-th chunk

1) Estimate the bandwidth share $\hat{x}[n]$ by

$$\frac{\hat{x}[n] - \hat{x}[n-1]}{T[n-1]} = k \times (w - \max(0, \hat{x}[n-1] - \tilde{x}[n-1] + w)) \tag{11}$$

2) Smooth out $\hat{x}[n]$ to produce filtered version $\hat{y}[n]$ by function S.
3) Quantize $\hat{y}[n]$ to the discrete video bitrate $r[n]$ by function Q.
4) Schedule the next download request via

$$\hat{T}[n] = \frac{r[n] \times \tau}{\hat{y}[n]} + \beta \times (B[n-1] - B_{min}) \tag{12}$$

The PANDA's bandwidth estimation mechanism has an additive-increase-multiplicative-decrease (AIMD) nature, which is similar to TCP congestion control [18]. The main difference is that congestion is indicated by the reduction of measured TCP throughput. If $\tilde{x} < \hat{x} + w$, where w is additive increase rate, it means that the estimated bandwidth is smaller than measured bandwidth or in other words, the link becomes undersubscribed. Thus, the client continuously increase the estimated bandwidth by kw per time unit (where k is probing convergence rate) in order to fully utilize the network resource. Otherwise, it indicates that the congestion has occurred, then the client should decrease the bandwidth estimation.

For smoothing, PANDA implements an exponential weighted moving average (EWMA) smoother of the form:

$$\frac{\hat{y}[n] - \hat{y}[n-1]}{T[n-1]} = -\alpha \times (\hat{y}[n-1 - \hat{x}[n]) \tag{13}$$

where $\alpha > 0$ is the convergence rate of $\hat{y}[n]$ towards $\hat{x}[n]$.

For quantization, they implement the dead-zone quantizer as following:

$$r[n] = \begin{cases} r_{up} & r[n-1] < r_{up} \\ r[n-1] & r_{up} < r[n-1] < r_{down} \\ r_{down} & otherwise \end{cases} \quad (14)$$

where r_{up} is upshift threshold and defined as $r_{up} = \max_{r \in R} r$ subject to $r < \hat{y}[n] - \Delta_{up}$. Similarly, r_{down} is downshift threshold where $r_{down} = \max_{r \in R} r$ subject to $r < \hat{y}[n] - \Delta_{down}$. The dead-zone $[r_{up}, r_{down}]$ reduce the bitrate switches between two adjacent levels, thus improving the stability of algorithm. In practice, $\Delta_{up} = 0$ and $\Delta_{down} = \epsilon \times \hat{y}$, where $0 < \epsilon < 1$ is a safety margin.

The scheduling step aims to achieve smoothed bandwidth $\hat{y}[n]$ at the end of the interval $\hat{T}[n]$, or $\hat{y}[n]$ is equal to $\frac{r[n] \times \tau}{\hat{T}[n]}$. Additionally, the algorithm would like to drive the buffer level toward to minimum reference level B_{min}, so that it adds the second term as in Equation (12), where β is to control the convergence rate.

C. Buffer-based adaptation

In this type of scheme, the client uses the playout buffer occupancy as a criterion to select the next segment bitrate.

1) BBA: BBA algorithm [19] defines a mapping function between video bitrate and buffer occupancy as Figure 5. In detail, at each time step, the client measures its buffer occupancy, which then is tuned to rate-buffer function $f(B)$ to get bitrate of the next video segment. A rate-buffer map function must satisfy the following constraints:

- Continuous functions of the buffer occupancy B.
- Strictly increasing in the region B: $R_{min} < f(B) < R_{max}$.
- Pinned at both ends, i.e., $f(0) = R_{min}$ and $f(B_{max}) = R_{max}$.

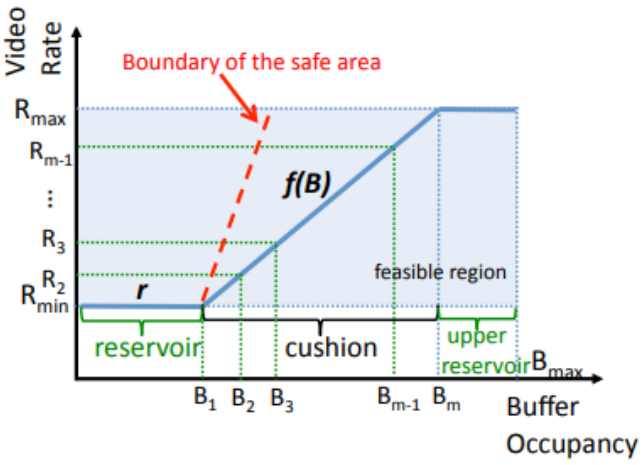


Fig. 5: The rate-buffer map used in BBA

To avoid interruption, the client needs to have at least several segments available in the buffer. The algorithm, therefore, introduces extra intervals: reservoir, cushion, and upper reservoir. When the buffer is filling up the reservoir, the algorithm always request the minimum video bitrate R_{min}. Once the reservoir is reached, the client chooses the next bitrate based on the rate-buffer map function $f(B)$ (cushion). When the buffer goes to upper reservoir, the maximum birate B_{max} is chosen.

BBA suffers from QoE degradation during long-term bandwidth fluctuations [4].

2) BOLA: BOLA algorithm [20] treats bitrate adaptation as a utility maximization problem. BOLA defines two performance metrics: *playback utility* $\bar{v}_N$ representing for quality of playback video and *playback smoothness* $\bar{s}_N$ that is expected fraction of time that is spent not rebuffering. The algorithm is designed to maximize the joint utility $\bar{v}_N + \gamma \bar{s}_N$ where γ is an input weight parameter for prioritizing playback utility versus the playback smoothness.

By solving the optimization problem, the optimal solution is to download the next segment at bitrate index $m^\star$ where $m^\star$ is the index that maximizes the ratio $\rho = (V v_m + V\gamma\tau - b[n])/R_m$. $b[n]$ denotes buffer occupancy in number of segments, which can be calculated by $b[n] = B[n]/\tau$. v_m is the utility of bitrate index m, which is non-decreasing function of bitrate. In BOLA, they choose the logarithmic utility function: $v_m = \ln(R_m/R_1)$. V is the control parameter which is defined as $V = \frac{b_{max} - 1}{v_1 + \gamma\tau}$.

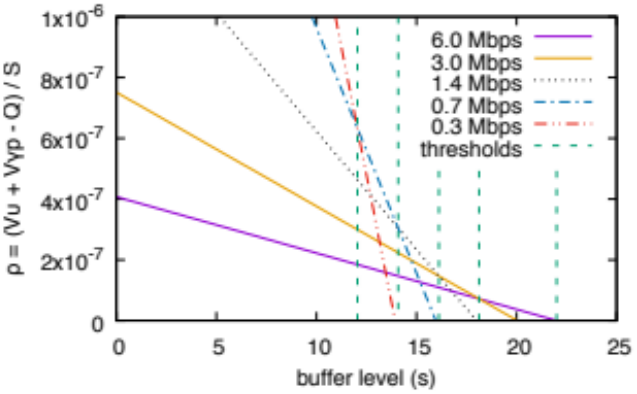


Fig. 6: The rate-buffer map used in BOLA

As shown in Figure 6, for each value of buffer occupancy, there is an optimal selection of bitrate that maxmimize the ρ ratio. This scheme somehow is equivalent to rate-buffer map of BBA. The main advantage is that the mapping function is determined with only two parameters V and γ.

So far, BOLA does not uses measured bandwidth to find optimal next segment's bitrate. It can be risky if the algorithm choose a bitrate level that is far bigger than measured bandwidth. Therefore, BOLA defines two schemes to safely switch up to higher bitrate level, which leads to two variants BOLA-U and BOLA-O. BOLA-O mitigates oscillations by introducing bitrate capping when switching to a higher bitrate. The segment bitrate is capped at the maximum level that is still

smaller than measured bandwidth. The second variant BOLA-U aims to maximize the utility by allowing selected bitrate can reach to the minimum bitrate level that is bigger than measured bandwidth. As the buffer occupancy will decrease when bitrate higher than measured bandwidth, BOLA-U can cause bitrate oscillations.

IV. Proposed Improvements

In multi-user scenarios, the aforementioned algorithms exhibit some drawbacks in terms of trade-off between efficiency, stability, and fairness. PANDA and FESTIVE are specially designed for multi-user cases. They are equipped with good schemes to perceive the fair-shared bandwidth, therefore, the algorithms achieve good performance in terms of fairness and stability in the steady state. However, due to the discrete nature of bitrate levels, efficiency of bandwidth usage is not high. For example, when two clients compete for the bottleneck of 10 Mbps so that the fair-shared bandwidth each client should receive is around 5 Mbps. Assume there are two bitrate levels of 4 Mbps and 6 Mbps. To maintain the fairness, the algorithms tend to choose the bitrate level of 4 Mbps for both clients, which leads to the waste of 2 Mbps bandwidth.

Buffer-based algorithms such as BOLA can be better than bandwidth-based ones in term of efficiency as they can choose the bitrate levels higher than measured bandwidth. However, they are not designed with dedicated tool to perceived the fair-shared bandwidth. This can cause bitrate fluctuation and unfairness between clients. This is because the fluctuation in measured bandwidth leads to the fluctuation of buffer occupancy, which ends up with fluctuation of bitrate.

Algorithm 3: Proposed Algorithm

for *each time step* n **do**

1. Estimate the bandwidth share $\hat{x}[n]$ by Equation 11.
2. Produce filtered version $\hat{y}[n]$ by Equation 13.
3. $m^\star[n] \leftarrow \arg\max_m (V v_m + V\gamma\tau - b[n])/R_m$.
4. **if** $m^\star[n] > m^\star[n-1]$ **then**
5. $\quad m' \leftarrow m$ such that $R_m < (1-\epsilon)\hat{y}[n]$.
6. $\quad$ **if** $m' < m^\star[n-1]$ **then**
7. $\quad\quad m^\star[n] \leftarrow m^\star[n-1]$
8. $\quad$ **else if** $m' > m^\star[n]$ **then**
9. $\quad\quad$ **if** $B_{[}n] < B_{optimal}$ **then**
10. $\quad\quad\quad m^\star[n] \leftarrow m'$
$\quad\quad$. **else**
11. $\quad\quad\quad m^\star[n] \leftarrow m'+1$
12. Download segment n at bitrate index $m^\star[n]$.
13. Schedule the next download as Equation 12.

To address the aforementioned problems of state-of-the-art algorithms, we propose some improvements. The main idea is instead of maintaining the fairness at each time step, we try to keep the fairness in long term. In previous example, where each client tend to choose the same bitrate to achieve the fairness, we give clients the abilities of choosing different bitrate from each other. Specifically, at time step n, client 1 can choose the bitrate of 6 Mbps and client 2 choose 4 Mbps. At another time step $n+i$, client 1 will take the bitrate 4 Mbps while the other selects 6 Mbps. In this scheme, the bandwidth of 10 Mbps is fully utilized at each time step and the fairness in the long term is still maintained. We will explain below why the proposed algorithm can achieve this target.

The proposed algorithm presented in Algorithm 3 is a combination of PANDA and BOLA. We take advantages of fair-shared bandwidth estimator and scheduler from PANDA as we consider them well-performed tools in multi-user scenarios. The bitrate selection algorithm is derived from BOLA with some modifications. The algorithm first calculates optimal bitrate index $m^\star[n]$ at time step n given by BOLA (line. 3). When switching to higher bitrate, the maximum bitrate index m' that is smaller than bandwidth is calculated as a reference. While in BOLA, the reference bitrate is determined by measured bandwidth, here, we use fair-shared bandwidth $\hat{y}[n]$instead as it is more stable. We also introduce ϵ, which controls the safety of algorithm. With larger ϵ, smaller m' is chosen.

In BOLA, when switching to a bitrate larger than bandwidth, the client always selects the bitrate index m' or $m'+1$. In our methods, we use the combination of the two options in an online decision. If current buffer occupancy is smaller than optimal buffer level, then we cap the next bitrate under fair-shared bandwidth to avoid bitrate fluctuation (line 10). Otherwise, when buffer is relatively large, we can choose the bitrate larger than fair-shared bandwidth to better exploit the resource (line 11).

The propose algorithm can maintain the long-term fairness in multi-user cases. The clients choosing bitrate larger than fair-shared bandwidth will experience buffer decreases, which ends up with switching down bitrate level. Meanwhile, buffer of clients choosing bitrate at level m' will increase, which leads to switch up bitrate. This scheme allows clients to achieve the target mentioned in the beginning of this section. Furthermore, by controlling the switching through the optimal buffer level, the algorithm can reduce the frequency of bitrate switches, which guarantees its stability.

V. Evaluation and Discussion

A. Experimental setup

In our experiments, we create a virtual network by Mininet [22] as shown in Figure 7. The network consists of a HTTP Server, and multiple clients, which are connected to the server via two routers. Links between server and Aggregation Router, and between Home Router and HAS Clients are local, which means that they are unbounded and zero delay. We configure the link between two routers to simulate network conditions.

HTTP Sever and Clients are implemented based on nghttp2 open source. A 400-second video is divided into temporal segments of 2 seconds ($\tau = 2s$). Each segment is encoded in 10 different bitrate levels: 459, 693, 937, 1270, 1745, 2536, 3758, 5379, 7861 and 11321 Kbps as in [5]. The buffer size for all algorithms is set to 30s. For fairness, the conventional player uses the same smoothing and quantizing functions as PANDA as recommended in [5].

Default parameters which are required for implementation of algorithms are listed in Table II. Each client joins the session at a random time instant and requests the lowest bitrate level for the first video segment in order to quickly fulfill its buffer.

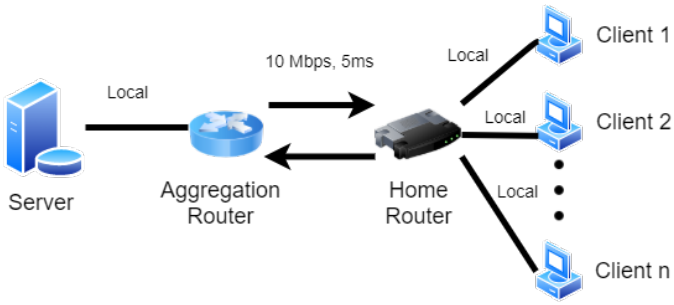


Fig. 7: Network topology used in simulation.

TABLE II: Parameters used in experiments

Algorithm	Parameter	Value
Conventional	α	0.2
	ϵ	0.15
FESTIVE	*targetbuff*	30s
	δ	12
PANDA	α	0.2
	ϵ	0.15
	k	0.14
	w	0.3
	β	0.2
	B_{min}	26s
BOLA	γ	2.5
Proposed method	γ	2.5
	ϵ	0.15
	$B_{optimal}$	28s

B. Experiment results

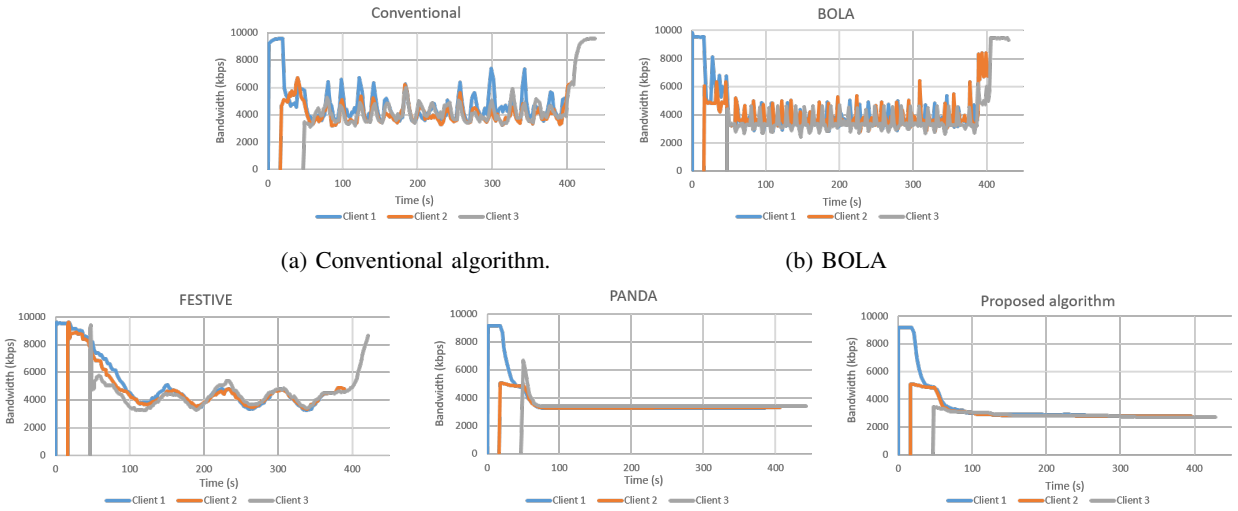
First, we consider a system of three clients completing for a bottleneck of 10 Mbps. Figure 8 shows the estimated shared bandwidth of proposed algorithm and the others. Conventional method in Figure 8a and BOLA in Figure 8b simply use measured bandwidth as estimate for the next video segment. Therefore, for both methods, estimated bandwidth experiences many fluctuations over whole session even when the total bandwidth remains unchanged of 10 Mbps. However, the Conventional method fluctuates less frequent than BOLA thanks to applying a filter to smooth out the estimate as shown in Equation 13. The side effect of the filter is to make the magnitude of variations larger. On the other hand, FESTIVE and PANDA as shown in Figure 8c and 8d, respectively are specially designed for estimating shared bandwidth in multi-user scenarios. FESTIVE estimates bandwidth by harmonic mean over last 20 samples, which is proved for the robustness against large outliers. Compared to the Conventional and BOLA, this estimation technique can mitigate short-term fluctuations. However, there still exists periodic oscillations in the long term over the session. PANDA exhibits the most accurate estimation, where the estimate of all clients eventually converge to the fair-shared bandwidth and remains at this value over the session. This is because PANDA implements the AIMD mechanism, which is similar to TCP congestion control, to probe the network. Our proposed method inherits the estimation mechanism from PANDA; no fluctuation, therefore, is observed in steady state of the session.

Figure 9 illustrates the chosen bitrate at each time step of algorithms over the video session. The chosen bitrate of Conventional algorithm in Figure 9a frequently fluctuates over the session due to fluctuations of estimated bandwidth. Despite being a buffer-based algorithm, BOLA still experiences short-term variations in chosen bitrate. This is due to the fact that fluctuations in bandwidth eventually leads to fluctuations in buffer level as formulated in Equation 3. Another reason is that the estimated bandwidth is also involved in bitrate selection of the algorithm. Besides, those two algorithms do not achieve a good fairness between clients in terms of chosen bitrate. To illustrate, with BOLA algorithm installed, Client 3 always selects lower bitrate than the others.

On the other hand, the chosen bitrate of FESTIVE in Figure 9c is in form of periodic square wave with some instant fluctuations. This form is derived from quantizing the sinusoidal periodic waveform of estimated bandwidth. The instant fluctuations are due to the fact that in some certain situations, bitrate selection is extremely sensitive with minor fluctuations of bandwidth. For example, when bandwidth is around 3700 kbps, the client tends to quantize the bandwidth to get the bitrate level of 2536 kbps. However, if the bandwidth increases slightly to 3800 kbps, the client will select another bitrate level, which is 3758 kbps. As shown in Figure 9c, most of the time, all clients choose the same bitrate, which proves for good fairness of the algorithm. In terms of efficiency, at some time step, for example at around 130-th second of the session, all clients select the bitrate of 2536 kbps, which is far smaller than fair-allocated bandwidth for each. PANDA in Figure 9d is strictly fair between users in steady state where all clients choose the same bitrate level, which remains unchanged over the session. However, similar to FESTIVE, PANDA does not exhibit good efficiency where all clients do not fully utilize the network resource.

Meanwhile, our proposed algorithm achieves a better trade-off between fairness and efficiency. As shown in Figure 9e, thanks to the estimation mechanism inherited from PANDA, the Proposed algorithm can precisely perceive fair-shared bandwidth for each user, which results in less fluctuations compared to the Conventional, BOLA and FESTIVE. The instant fluctuations as seen in FESTIVE and BOLA is also removed by using an optimal buffer level to avoid unnecessary switches. Our algorithm is only less stable than PANDA but outperforms PANDA in terms of efficiency where total chosen bitrate at each time step is close to the total bandwidth.

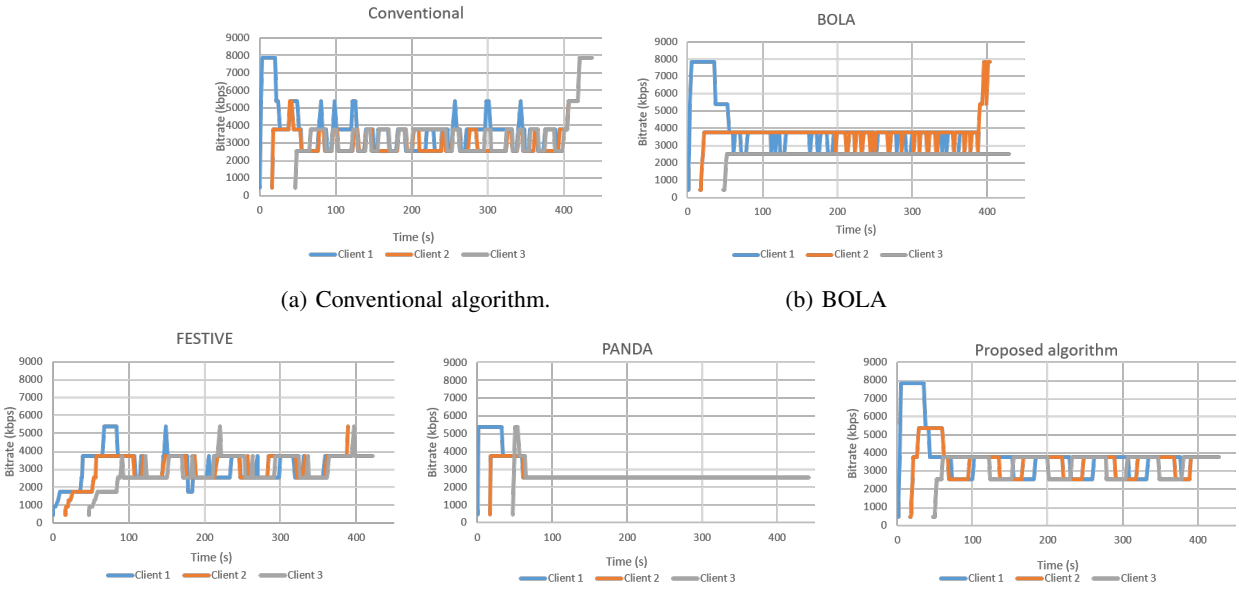
For a clearer comparison, we show in Figure 10 the average bitrate of each client with each algorithm. The Conventional and BOLA better exploits the network resource where total



(a) Conventional algorithm. (b) BOLA

(c) FESTIVE (d) PANDA (e) Proposed algorithm.

Fig. 8: Comparison of estimated bandwidth between adaptive algorithms.



(a) Conventional algorithm. (b) BOLA

(c) FESTIVE (d) BOLA (e) Proposed algorithm.

Fig. 9: Comparison of chosen bitrate between adaptive algorithms.

Algorithm	Client 1 (kbps)	Client 2 (kbps)	Client 3 (kbps)
Conventional	3636.415	2977.63	3365.765
BOLA	3703.22	3581.93	2440.97
FESTIVE	2818.05	2878.85	2825.855
PANDA	2632.112	2678.365	2578.485
Proposed	**3255.09**	**3178.18**	**3115.585**

Fig. 10: Average bitrate of 3 users.

average bitrate over all clients is almost equal to total bandwidth of 10 Mbps. However, they do not fair between clients. To illustrate, with BOLA installed, the average bitrate of Client 1 and Client 2 (3703 kbps and 3581 kbps, respectively) are far higher than that of Client 3 with merely 2440 kbps. On the other hand, FESTIVE and BOLA achieve better fairness where all clients witness approximate bitrate values. However, they do not fully utilize the allocated resource where the average bitrate of each client is much smaller than fair-shared bandwidth. The proposed algorithm takes advantages from those two groups. It gives the higher average birate than FESTIVE and PANDA and better fairness between users than the Conventional and BOLA.

In the second experiment, we extend the system to five clients compete for the bottleneck. We compare the efficiency of the algorithms by the average bitrate over all clients and the fairness by Jain's fairness index as formulated in [21]. The closer the index is to 1, the better the fairness is. Figure

Algorithm	Average bitrate (kbps)	Jain's fairness index
Conventional	3404.05	0.993437
BOLA	3357.33	0.970158
FESTIVE	2835.20	0.999909
PANDA	2630.26	0.999759
Proposed	**3200.98**	**0.999678**

Fig. 11: Efficiency and Fairness of the algorithms in case of 3 users.

Algorithm	Average bitrate (kbps)	Jain's fairness index
Conventional	1985.147	0.996219
BOLA	2419.45	0.936587
FESTIVE	1958.42	0.999637
PANDA	1763.025	0.999930
Proposed	**2009.12**	**0.999020**

Fig. 12: Efficiency and Fairness of the algorithms in case of 5 users.

11 and Figure 12 show the efficiency and fairness of the algorithms in case of 3 users and 5 users, respectively. In both cases, the proposed algorithm exploits the bandwidth resource more efficiently than FESTIVE and PANDA by getting higher average bitrate over clients. Our algorithm is also much fairer than the Conventional and BOLA but not far away from PANDA and FESTIVE, which is proved by Jain's fairness index.

VI. Conclusions

While a simple rate adaptation algorithm can perform well with a single client, mutli-user scenarios could lead to unpredictable problems. Two sources of bias have been introduced in this work namely initial condition bias and bitrate selection bias. In this work, we also take a look into two groups of state-of-the-art algorithms and investigate their performance in multi-user scenarios. The first group including the Conventional algorithm, which is widely used in commercial players, and BOLA is not well designed to work with multiple clients. They, despite better exploiting the bandwidth resource, show the poor fairness between clients. On the other hand, FESTIVE and PANDA, which are equipped with effective tools to perceive the shared bandwidth, are fairer in terms of chosen video bitrate between clients. However, they do not efficiently leverage network resource. Our algorithm is proposed to address the problems witnessed in those two groups. We use the same bandwidth estimation mechanism as PANDA and inherit the bitrate selection from BOLA with some improvements for better adaptation in multi-user scenarios. Our algorithm is proved for a better trade-off between efficiency and fairness compared to the reference ones.

References

[1] T. C. Thang, Q.-D. Ho, J.-W. Kang, and A. T. Pham, "Adaptive Streaming of Audiovisual Content using MPEG DASH," IEEE Trans. Consum. Electron., vol. 58, no. 1, pp. 78–85, Feb. 2012

[2] S. Tullimas, T. Nguyen, R. Edgecomb, and S.-C. Cheung, "Multimedia streaming using multiple tcp connections," ACM Trans. Multimedia Comput. Commun. Appl (ACM TOMCCAP), vol. 4, no. 2, pp. 1–20, May 2008.

[3] A. C. Begen, T. Akgul, and M. Baugher, "Watching video over the web: Part 2: Applications, standardization, and open issues," IEEE Internet Computing, vol. 15, no. 3, pp. 59–63, Apr. 2011

[4] Abdelhak Bentaleb, Bayan Taani, Ali C. Begen, Christian Timmerer, and Roger Zimmermann, "A survey on bitrate adaptation schemes for streaming media over HTTP," IEEE Communications Surveys & Tutorials, vol. 21, no. 1, pp. 562–585, 2018.

[5] 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.

[6] J. Jiang, V. Sekar, and H. Zhang, "Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE," in Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 97–108.

[7] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service," SIGCOMM Comput. Commun. Rev., vol. 44, no. 4, pp. 187–198, Aug. 2014.

[8] 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.

[9] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hossfeld, and P. TranGia, "A Survey on Quality of Experience of HTTP Adaptive Streaming," Communications Surveys Tutorials, IEEE, vol. 17, no. 1, pp. 469-492, 2015.

[10] Ricky K. P. Mok, Edmond W. W. Chan, and Rocky K. C. Chang, "Measuring the Quality of Experience of HTTP Video Streaming," in IFIP/IEEE International Symposium on Integrated Network Management and Workshops, Dublin, Ireland, May. 2011.

[11] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang," QDASH: A QoE-aware DASH system," in Proc. 3rd Annu. ACM Conf. MMSys, Chapel Hill, NC, USA.

[12] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate Adaptation for Adaptive HTTP Streaming," in Proceedings of the Second Annual ACM Conference on Multimedia Systems, ser. MMSys '11. New York, NY, USA: ACM, 2011, pp. 169–174.

[13] S. Akhshabi, S. Narayanaswamy, Ali C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptive video players over HTTP. Signal Processing: Image Communication, 27:271–287, 2012.

[14] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. Confused, timid, and unstable: Picking a video streaming rate is hard. In Proc. 2012 ACM conference on Internet measurement, 2012.

[15] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj. Rate Adaptation for Adaptiv HTTP streaming. In Proc. ACM Multimedia Systems Conference (MMSys'11), pages 169–174, San Jose, CA, USA, February 2011.

[16] Harmonic mean. http://en.wikipedia.org/wiki/Harmonic_mean

[17] N. Cranley, P. Perry, and L. Murphy. User perception of adapting video quality. International Journal of Human-Computer Studies, 2006.

[18] V. Jacobson. Congestion avoidance and control. In Symposium proceedings on Communications architectures and protocols, SIGCOMM '88, pages 314–329, New York, NY, USA, 1988. ACM.

[19] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service," SIGCOMM Comput. Commun. Rev., vol. 44, no. 4, pp. 187–198, Aug. 2014.

[20] 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.

[21] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer system. Technical Report, DEC, 1984.
[22] Mininet http://mininet.org/
[23] Nghttp2 https://nghttp2.org/