Multimedia Systems (2018) 24:509–529
https://doi.org/10.1007/s00530-018-0588-7

REGULAR PAPER

SABA: segment and buffer aware rate adaptation algorithm for streaming over HTTP

Waqas ur Rahman[1] · **Kwangsue Chung**[1]

Received: 20 November 2016 / Accepted: 16 March 2018 / Published online: 21 March 2018

Abstract
Adaptive streaming allows for dynamic adaptation of the bitrate to varying network conditions, to guarantee the best user experience. Adaptive bitrate algorithms face a significant challenge in correctly estimating the throughput, as the throughput varies widely over time. The current throughput estimation methods cannot distinguish between throughput fluctuations of different amplitude and frequency. In this paper, we propose a throughput estimation method that accurately estimates the throughput based on previous throughput samples. It is robust to short term and small fluctuations, and sensitive to large fluctuations in throughput. Furthermore, we propose a rate adaptive algorithm for video on demand (VoD) services that selects the quality of the video based on the estimated throughput and playback buffer occupancy. The objective of the rate adaptive algorithms is to guarantee high video quality to improve user experience. The proposed algorithm dynamically adjusts the quality level of the video stream. The proposed method selects high quality video segments, while minimizing the risk of playback interruption. Furthermore, the proposed method minimizes the frequency of video rate changes. We show that the algorithm smoothly switches the video rate to improve user experience. Furthermore, we determine that it efficiently utilizes network resources to achieve a high video rate; competing HTTP clients achieve equitable video rates. We also confirm that variations in the playback buffer size and segment duration do not affect the performance of the proposed algorithm.

Keywords HTTP-based video streaming · Quality of experience · Video rate adaptation algorithm · Video streaming scheme

1 Introduction

High-speed broadband networks and improvements in display technology of various devices (e.g., smart phone, table PC, and personal media player) have enabled video streaming to become the most popular application over the Internet. Most commercial video streaming services run over HTTP to provide high quality video. The majority of these services use rate adaptive algorithms that adapt the video quality, by observing the available throughput or playback buffer occupancy.

HTTP-based video streaming solutions provide multiple representations (e.g., different bitrate/quality) of the same content, and divide these representations into small segments. The content is stored at the server, and the rate adaptive algorithm at the client decides which segment to download next. The algorithms try to maximize the quality of the video, by meeting conflicting objectives in such a way as to improve the user's viewing experience. Some of the potential objectives include selecting a set of video bitrates that are the highest feasible, avoiding needless video bitrate switches, and preserving the buffer level to avoid interruption of playback [1–5]. Simply maximizing the video rate risks rebuffering, whereas simply minimizing rebuffering leads to low video quality.

The estimation of throughput plays an important role in the selection of the next segment. Several methods have been proposed to estimate the throughput of the upcoming segment [6–8]. HTTP clients make an estimate of the future throughput from past observations to select the video rate for the next segment [9–11]. Accurate estimation of the

Communicated by M. Claypool.

✉ Kwangsue Chung
kchung@kw.ac.kr

[1] Department of Electronics and Communications Engineering, Kwangwoon University, 447-1 Wolgye-Dong, Nowon-Gu, Seoul, Republic of Korea

throughput becomes an important challenge for the client. Inaccurate estimation may lead to selecting video bitrates that degrade user experience. To select video bitrates, the rate adaptive algorithms add playback buffer occupancy as an adjustment parameter on top of throughput estimation [7, 12–14].

In this paper, we first propose a throughput estimation algorithm with detection and estimation method. The proposed throughput detection method can distinguish between different types of fluctuations in the throughput. Based on the result of throughput detection, the estimation method offers a stable response to short term fluctuations, and is sensitive to persistent large fluctuations. We then propose a rate adaptation algorithm that dynamically selects the video bitrates based on the estimated throughput and the playback buffer occupancy. The objective of the proposed algorithm is to improve the viewing experience of the users. The algorithm streams high quality video, while avoiding playback interruption. The proposed adaptation algorithm minimizes the video rate changes, and makes sure that the video rate changes smoothly to improve the user experience. We perform experiments to show that irrespective of the buffer size and segment duration, the proposed algorithm improves user experience. Additionally, in a multi-client environment, we show that the proposed scheme efficiently utilizes network resources and that the HTTP clients achieve equitable video rates.

The rest of this paper is organized as follows: Sect. 2 offers an overview of HTTP adaptive streaming, and reviews the existing video streaming algorithms. Section 3 presents the proposed throughput estimation algorithm. Section 4 explains our throughput and buffer-based rate adaptive algorithm. Section 5 provides simulation results. Finally, Sect. 6 concludes the paper.

2 Overview of HTTP streaming

2.1 HTTP adaptive streaming

HTTP adaptive streaming works by monitoring network in real time, and by adjusting the quality of the video stream accordingly, without resetting the TCP connection. Figure 1 shows the basic model of adaptive HTTP streaming, which requires the server to store multiple versions of the multimedia content. At the server side, the content annotation module provides information about the characteristics of the stored multimedia content. The client initiates a request for information about the stored content, which is known as metadata. In response to the request from the client, the server sends the metadata to the client. The media preparation module provides tools for encoding and encapsulation, so that the content can be presented and efficiently delivered to the client in the correct format. On the client side, the scheduling module is responsible for scheduling the download of upcoming segments. During the download of the segments, the bandwidth estimation module estimates the throughput. The adaptation module selects a suitable bitrate depending on the received metadata and system conditions, such as the throughput and occupancy of the playback buffer. Once a segment is downloaded, it is temporarily stored in the playback buffer that feeds the player's decoder.

2.2 Related work

Currently, several methods have been proposed to estimate the throughput. Segment throughput is calculated as the ratio of segment size divided by the time it takes to download the segment. In the simplest way, measured segment throughput can be used as the throughput estimate of the next segment [10]. However, due to short term fluctuations, the throughput estimate calculated in this way will result in high frequency of fluctuations. Akhshabi et al. [9] evaluate the performance of Microsoft Smooth Streaming and Netflix player using the running average of the throughput of several

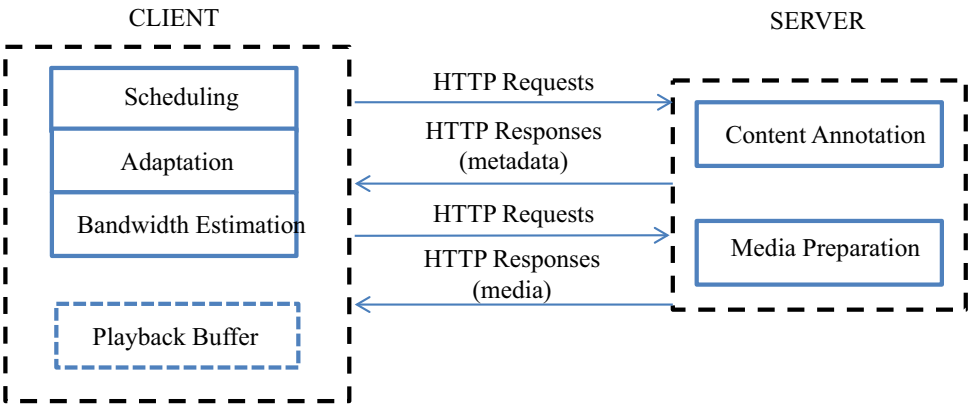


Fig. 1 HTTP streaming architecture

segments as the estimated throughput. The method performs well under persistent throughput variations. Ran et al. [6] use the median of the throughput of the last several segments to estimate the throughput of the next segment. Rahman et al. [7] show that the McGinely dynamic indicator offers a stable response to the throughput fluctuations, while maintaining a stable playback buffer. The moving average technique [7] is accurate in slow throughput variation, but reacts late to sudden variations in the throughput. The VLC media player [8] uses the averages of all previous throughputs as the estimated throughout to download the next segment. The method responds slowly to the actual throughput variations, which increases the risk of buffer underflow. Jiang et al. [15] use the harmonic mean of the throughput of the last 20 downloaded segments to estimate the throughput of the next segment. The outliers of the throughput do not influence the estimated throughput, but the method shows delay during persistent throughput variations. Many commercial clients estimate throughput of the next segment by taking the moving average of the previously downloaded segments [4]. The method is late to respond to large variations in the throughput. The throughput estimation methods proposed so far either have an aggressive or a stable response to the throughput fluctuations. As the rate adaptive algorithms select video rates based on the estimated throughput, an aggressive response results in higher number of throughput fluctuation. On the other hand, the stable response increases the risk of buffer underflow and inefficient utilization of the bandwidth. The proposed estimation method can differentiate between small and large fluctuations in the throughput based on variations in the frequency and amplitude of the network throughput. Based on the behavior of the network throughput, the estimation method decides whether to offer an aggressive or stable response.

References [9–11] propose rate adaptation algorithms that select the video rates based on the estimated throughput. These algorithms have been found to be either slow to converge to optimum solution, resulting in high frequency of video bitrate switching, or to result in a higher number of playback interruptions. In an unstable environment, inaccurate throughput estimation results in the degradation of the user experience.

Many methods have been proposed to incorporate information about the playback buffer in selecting the video rate. Miller et al. [12] propose a method that divides the buffer into multiple predefined thresholds (B_1, B_2, B_3, B_{max}) where ($B_1 < B_2 < B_3 < B_{max}$), and takes different decisions to select the video rates when the buffer level remains in different ranges. The algorithm does not dynamically adjust the buffer threshold as the segment duration and segment sizes of a VBR encoded video stream vary. Rahman et al. [7] propose an algorithm that intelligently selects the video bitrates based on the estimated throughput and buffer occupancy, by dynamically selecting buffer thresholds based on the sizes of the upcoming segments. The algorithm does not adjust the buffer thresholds as the buffer size and segment duration varies. Huang et al. [16] propose a video rate adaption algorithm that selects the video rate by only observing the client's playback buffer. The authors observe that due to the highly variable network dynamics, especially in the case of wireless networks, it is not easy to estimate throughput. Hence, they limit the throughput estimation to the initial stage. To handle the variation of segment sizes, the method directly maps the buffer occupancy to the segment size, and increases or decreases the video rate as the buffer builds up or drains, respectively. Furthermore, in deciding to switch the video quality, the algorithm considered the sizes of the upcoming segments. Rahman et al. [17] propose an algorithm that selects the video rates only based on the buffer occupancy by exploiting the variation of sizes of the upcoming segments. The algorithm maps the buffer occupancy to the video rate rather than the segment size, as mapping of the buffer level to the segment size results in a higher frequency of switches. Authors in [18] propose a user-centric streaming algorithm for H.264/SVC DASH streaming which adapts its quality according to the playback buffer level only. Dubin et al. [19] provides a rate adaptive algorithm that uses a double Exponential Moving Average (EMA) algorithm. The video quality is selected based on both playback buffer level estimation and throughput estimation. It is designed for multicast networks but the authors showed that it provides a stable performance under both multicast and unicast conditions. Authors in [20] propose the algorithm that adapts the video quality based on crowdsourcing data generated by users of a professional service. In addition, the authors integrate crowd information with the existing algorithms and show that read-world data can improve the performance of existing algorithms.

The proposed algorithm selects the video rate based on throughput estimation and playback buffer occupancy. The algorithm uses two video rate maps to select the video rate of the upcoming segments. To increase the video rate, the algorithm uses a rate map based on a concave function to aggressively increase the video rate in order to efficiently utilize the available throughput, and uses a linear function to conservatively decrease the video rate to avoid playback buffer interruption. The algorithms proposed so far do not dynamically adjust the video rate maps as the playback buffer sizes, segment durations and available set of video rates vary. The proposed algorithm dynamically adjusts the video rate maps as the buffer size of the client, segment duration and available video rates of the video stream vary.

3 The throughput estimation method

The rate adaptive algorithms strive to maximize the user experience by meeting conflicting video quality objectives. Some of the potential objectives include selecting the highest feasible set of video bitrates, avoiding needless video bitrate switches, and avoiding interruption of playback. The rate adaptive algorithms select the next segment on the basis of the estimated throughput. Therefore, it is important for the throughput estimation method to have a stable response to small variations in the throughput, to minimize unnecessary fluctuations in the video rate, and to react quickly to large fluctuations to minimize the risk of playback interruption due to buffer underflow.

3.1 Throughput detection method

The throughput detection method should be able to distinguish between different types of network conditions. To differentiate between fluctuations of different amplitude and frequency in the throughput, we calculate log return. The log return shows the extent of variability of the throughput in relation to the average throughput. Let $T(i)$ and $\overline{T}(i)$ denote the instant and average throughput, respectively, observed at the download of segment i. We calculate the log return ρ using:

$$\rho = \log \frac{\left|T(i) - \overline{T}(i)\right|}{\overline{T}(i)}, \tag{1}$$

where

$$\overline{T}(i) = \frac{\sum_{j=i-n-1}^{i-1} T(j)}{n}. \tag{2}$$

A high value of log return means that the difference between $T(i)$ and $\overline{T}(i)$ is significantly high, due to large fluctuations in the throughput. The client must react quickly to the large fluctuations in throughput. A smaller value of log return means a small fluctuation in the throughput or a short-term fluctuation. The client should offer a smooth and stable response to small fluctuations in throughput.

3.2 Throughput estimation method

After the throughput detection method detects the type of network condition, the client estimates the throughput. We estimate the throughput in (3) using the weighted average of the throughput observed over the last n segments.

$$T^E(i) = \sum_{j=i-n-1}^{i-1} \sum_{k=1}^{n} p(k) \times T(j). \tag{3}$$

The weighted factor p in (4) depends on the type of throughput fluctuation. If the throughput has large persistent variations, the throughput in recent times has higher weight, which makes the estimation quickly adjust to the actual throughput. We use the exponential function to give higher weight to recent throughput. In the case of small variations, we use the mean of the past throughput observations to provide stable estimation.

$$p = \begin{cases} \dfrac{\alpha^{k^2}}{\sum_{l=1}^{n} \alpha^{l^2}} & \text{if } \rho \geqslant \lambda \\ \dfrac{1}{n} & \text{else} \end{cases}, \tag{4}$$

If the value of $\rho \geq \lambda$, the exponential function is selected to make sure more recent throughput has higher weight; otherwise, the method uses mean of the past observations as the weighted factor. We perform experiments to select the value of λ to detect the type of throughput fluctuations. To this end we use rectangular waveform, shown in Fig. 2.

$A_{\max}$ and $A_{\min}$ denote the maximum and minimum values of throughput respectively and Δ represents their difference. L represents the duration of the framework and D is the proportion of time when the throughput is $A_{\max}$. We observe the response of the throughput estimation method by varying value of λ as shown in Figs. 3 and 4.

In the first experiment, Δ is varied while keeping $A_{\max}$ equal to 3000 kbps and varying $A_{\min}$. We vary the values of Δ from 250 to 1500 kbps. We want to observe how the proposed estimation method reacts to small and large throughput variations. The objective is to select the value of λ that reacts quickly to the throughput variations to efficiently utilize the bandwidth and minimize the risk of buffer underflow. We vary the value of λ from 0.05 to 0.125. The value of L and D are kept to 60 s and 0.5 respectively. Figure 3 shows that for Δ equal to 250 kbps, setting value of λ equal to

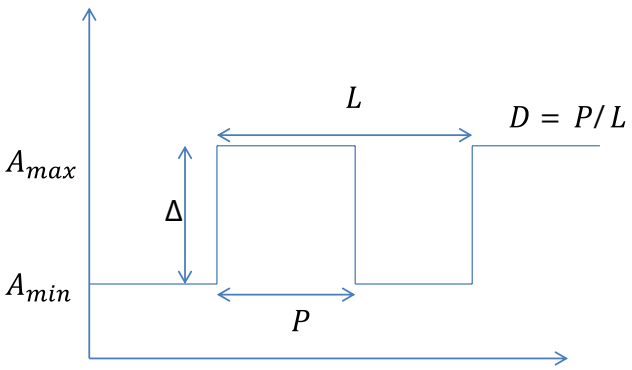
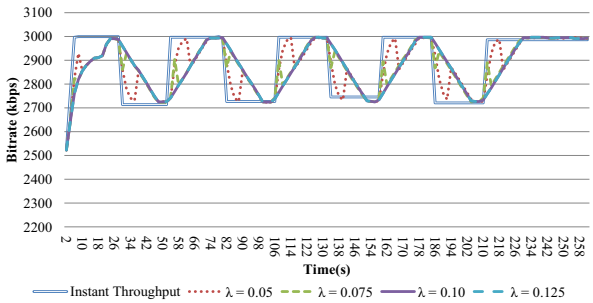
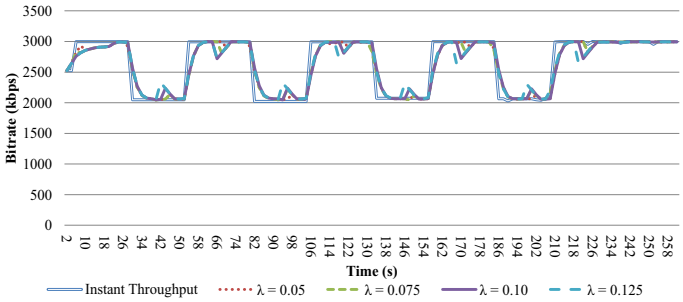


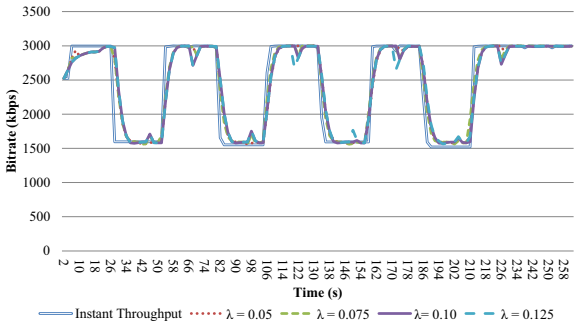
Fig. 2 A rectangular waveform throughput trace



(a) Δ = 250 kbps

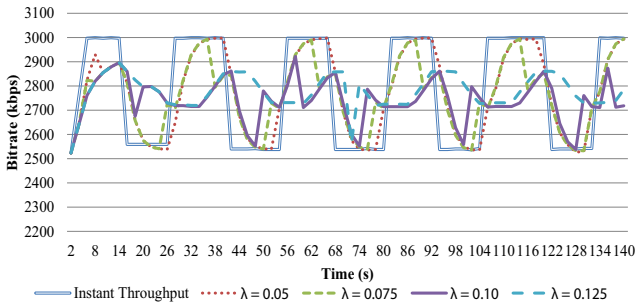


(b) Δ = 1000 kbps

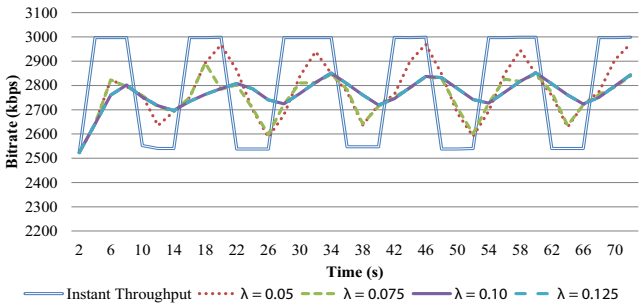


(c) Δ = 1500 kbps

Fig. 3 Performance of the proposed scheme as the amplitude of fluctuations changes



(a) L = 25 sec



(b) L = 16 sec

Fig. 4 Performance of the proposed scheme as the frequency of fluctuations changes

0.05 results in aggressive response to the fluctuations in the throughput. As the value of λ increases, the response to the throughput fluctuation becomes conservative. As the value of Δ increases, the throughput estimation method should be able to react quickly to the fluctuations in the throughput to efficiently utilize the throughput or avoid drop in buffer occupancy due to the throughput overestimation. Figure 3 shows that for large values of Δ, the proposed method reacts quickly to the changes in throughput for all value of λ.

In the next experiment, L is varied while keeping $A_{\max}$ equal to 3000 kbps. The objective is to select the value of λ that stabilizes short-term fluctuations to minimize the unnecessary video rate changes. The value of Δ and D are kept to 450 kbps and 0.5 respectively. In Fig. 4, we vary the frequency of throughput variation. We vary the value of L to observe how the proposed algorithm reacts to long and short-term fluctuations. Figure 3 shows the response of the proposed estimation method as for the value of L equal to 60 s. Figure 4 shows that as the value of L is reduced to 16 s, larger value of λ results in a stable response where as a smaller values of λ show an aggressive response. As the value of λ is increased above 0.125, we observed that the

proposed estimation method becomes slow to react to large fluctuations in throughput; therefore, we plotted the results only for the value of λ up to 0.125. The reason behind keeping a difference of 0.05 between the plotted values of λ is that the smaller values resulted in overlapping of the curves which makes the plot reading difficult.

The rate adaptation algorithms select the video rates on the basis of the estimated throughput and the playback buffer occupancy. As explained earlier, the major factors that affect the user experience include the average video rate, playback interruption and frequency of video rate changes. The purpose of the proposed throughput estimation method is to assist the rate adaptive algorithm proposed in the next section to select the video rates. Based on the experiments results shown in Figs. 3 and 4, we select λ equal to 0.1 for the proposed scheme. We observe that when λ is selected equal to 0.1, the proposed method reacts quickly to large fluctuations in the throughput; whereas, in case of small or short-term fluctuations in the throughput, the proposed method reacts conservatively to stabilize the estimated throughput. When the value of ρ is less than 0.1, the proposed technique uses the mean of throughputs observed over the previous n segments. As the value of ρ increases above 0.1, the proposed scheme selects the exponential function to react quickly to the variations in throughput.

To evaluate the performance of the proposed method, we implement the throughput estimation method in the network simulator, ns-3. The server offers discrete bitrates from 400 to 3000 kbps, with a step size of 200 kbps. The duration of each segment is 2 s, and the client starts playback after a segment has completely downloaded. Many commercial clients [21] use the previous 10 samples to estimate the throughput; therefore, we set the value of n equal to 10 for the proposed method. We set the value of α equal to 0.9 to react quickly to the large fluctuations in throughput. The video bitrate is determined by selecting the highest video rate that is less than the estimated throughput.

Figure 5 uses the proposed throughput estimation method to estimate the throughput. The proposed scheme shows a smooth response to small fluctuations; it is able to detect small variations, and offers a stable response to small fluctuations. Furthermore, the proposed scheme reacts quickly to large drop in the throughput. The rate adaptive algorithms select the video rates based on the estimated throughput; therefore, a late response to a large drop increases the risk of interruption in the playback. The proposed scheme is able to detect a large drop, and in order to quickly react to the drop in throughput, estimates the throughput exponentially. As the proposed scheme shows a smooth response to small fluctuations, it reduces the video bitrate switches. Figure 5 shows that the client does not change the video rate during small throughput fluctuations. As the throughput suddenly drops, the video rate drops quickly to avoid buffer underflow. As the client selects the highest video rate that is less than the estimated throughput, the buffer level increases gradually. When the throughput suddenly drops, the buffer initially drops, but as the client quickly drops the video rate, the buffer level stabilizes.

To further evaluate the performance of the proposed scheme, we compare with the method that estimates throughput by dividing the download size by the download time and passing it through moving average filter [21].

Figure 6 compares the performance of the proposed scheme with the moving average throughput estimation method. Figure 6a shows that unlike proposed scheme, moving average method reacts slowly to the actual throughput. This not only results in underutilization of the available throughput when the throughput increases but also risks

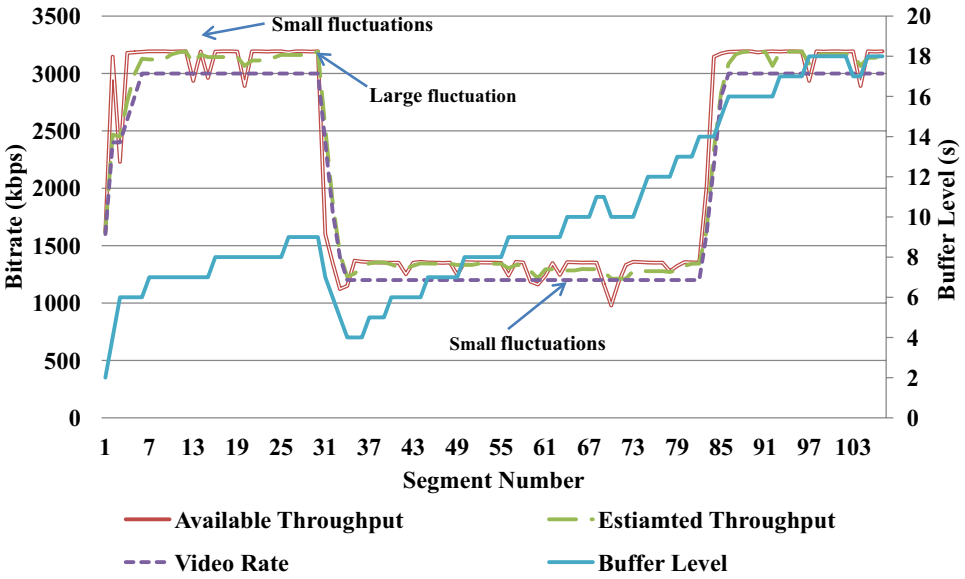
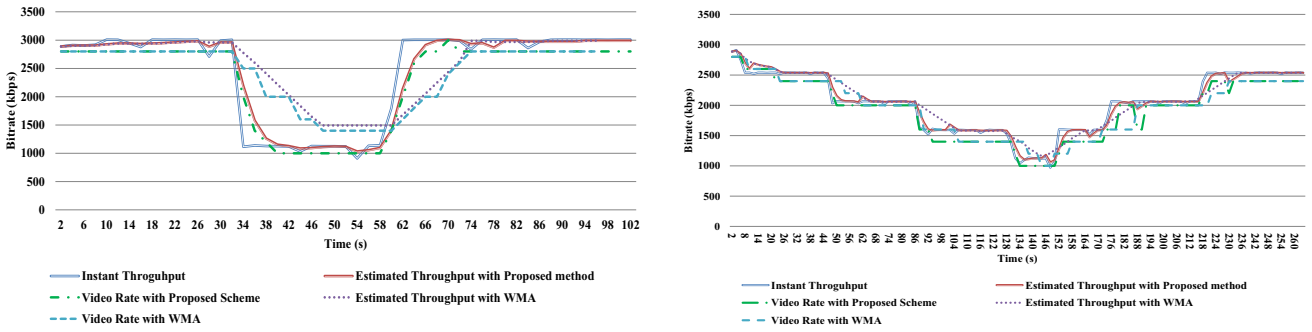


Fig. 5 Throughput estimation under a predetermined network scenario



(a) Small and large throughput fluctuations network scenario

(b) Gradual drop and gradual increase network scenario

Fig. 6 Performance comparison between proposed and moving average throughput estimation method

playback interruption due to buffer underflow when the throughput suddenly drops. In case of a gradual drop and increase in the throughput, the proposed scheme accurately estimates the throughput. The weighted average method reacts slowly to the actual throughput which results in inefficient utilization of the throughput.

4 Proposed algorithm

In this section, we propose a segment and buffer aware (SABA) rate adaptive algorithm. The proposed rate adaptive algorithm selects the video rate based on the estimated throughput and the playback buffer level. Video rates and rebuffering events are important factors for improving the user experience. In addition, frequent video rate switching has been found to annoy the viewer. The main goal of the proposed algorithm is to adaptively select a video rate from a set of video rates $R = \{R_1, R_2, R_3, \ldots, R_n\}$, to optimize the viewing experience.

4.1 System model

The video stream is segmented into n segments, each containing τ seconds of playback, and stored at the server side. Each segment is available in multiple bitrates. The set of representations available for the video stream is denoted by R. The client dynamically selects a video rate from the set R. The client selects the kth video rate, R_k, from the set R for each segment, to adapt the video according to the estimated throughput and playback buffer. R_{min} and R_{max} are the representations with the lowest and highest video rate in the set R.

The presented work uses a serial segment fetching method to download segments, which requests the next segment after the current segment has completely downloaded. Once the current segment has completely downloaded, it adds data of τ seconds to the buffer. After the first segment has downloaded, the client starts playing the video.

4.2 Adaptive bitrate algorithm

Algorithm 1 provides the algorithm's pseudo-code. We invoke Algorithm 1 immediately after segment i-1 is downloaded. The algorithm selects the representation for the download of the next segment i. The algorithm considers the buffer level and throughput together.

To download the first segment, the client always selects the minimum available video rate, R_{min}. There are two reasons for selecting R_{min} as the video rate of the first segment. First, as the buffer builds up from being empty, it carries little information with which to select a video rate. We consider a conservative approach at the start, and as the buffer gradually increases, we start taking more risk in selecting the video rate. Secondly, downloading the segment with the smallest video rate reduces the initial delay. Waiting time impairment such as initial delay is of considerable interest in HAS systems [22].

To select the kth video rate, two conditions should be satisfied. Firstly, the selected video rate should be less than the estimated throughput. To avoid depletion of the buffer, the first condition makes sure that the selected video rate is below the estimated throughput. The throughput is estimated using the estimation method described in Sect. 3. Secondly, for a client to select the kth video rate, the buffer level should be higher than the threshold, B_k. The reason behind this condition is to reduce the risk of playback interruption due to buffer underflow in case the throughput is estimated inaccurately.

The video rate for the next segment is selected on a segment-by-segment basis. We consider the buffer dynamics when the segment has completely downloaded. Let $B(i-1)$ be the buffer level at the end of the download of segment $i-1$. $B(i)$ is then given by:

$$B(i) = B(i-1) + \tau - \left[\tau \times \frac{R_k(i)}{T(i)}\right], \tag{5}$$

where $R_k(i)$ is the kth video rate from the set R, and $T(i)$ is the video throughput observed during the download of segment i. Equation (5) shows that a video rate greater than the throughput drains the playback buffer.

Algorithm 1: Adaptation Algorithm

```
B_k: Buffer threshold to select the k^th video rate
B(i): Buffer level at the download of the i^th segment
R_prev : Video rate selected for the previous segment
R_next : Video rate selected for the next segment
n: Number of available video rates
i: Index of the next segment
k: Index of the current video rate

//Select the lowest available video rate
if B(i-1) < B_min then
        R_next = R_min

//Select the highest available video rate
else if B(i-1) > B_n then
        R_next = R_max

//Decrease the video rate
else if R(i-1)≠R_min && B(i-1)< B̄_k && R_k(i-1)>T^E(i) then
        for j= k-1 to 1
          if R_j(i-1)<T^E(i) && B(i-1)>Bl_j
              R_next ← R_j
              break
          end if
          j ← j-1
        end for

//Increase the video rate
else if R_k(i-1)≠R_max && R_k+1<T^E(i) && B(i-1)>Bc_k+1 then
        for j=k+1 to n
          if B(i) > Bc_j && T^E(i) > R_j
              R_next ← R_j
          end if
          j ← j+1
        end for

else
        R_next = R_prev
```

We use mathematical buffer models to select the thresholds, B_k, used to select the video rates. The buffer models are based on a mathematical function that restricts the video bitrates based on the buffer occupancy level. We use two buffer models to select the video rates based on the buffer occupancy levels. Figure 7 shows that we use buffer models based on concave and linear functions to increase and decrease the video rates as the buffer occupancy increases and decreases, respectively. When the buffer level increases, we use the concave buffer model to restrict the video rates; and when the buffer level decreases, we switch to the linear buffer model. In Fig. 7, Bc_k and Bl_k are the buffer occupancy thresholds to select the kth video rates when the client uses the concave and linear functions, respectively. When the buffer occupancy increases, the client uses the threshold Bc_k to select the video rate; and when the concave wave suggests a lower video rate, the client switches to the threshold Bl_k to pick the video rate. It selects the threshold, B_k, to select the kth video rate as:

$$B_k = \begin{cases} Bc_k & \text{when the client decides to increase the video rate} \\ Bl_k & \text{when the client decides to decrease the video rate.} \end{cases} \tag{6}$$

The buffer model based on the concave function to calculate the video rate restriction γ for a segment s_i [23], is given by:

$$\gamma(s_i) = \alpha \times \log_b(\delta_i \times c), \tag{7}$$

where $i \in [1,N]$ represents the segment number, δ_i the buffer occupancy level, and α, b, c are the predefined parameters to fine-tune the model. The buffer occupancy level ranges from 0 to 1; 0 means that the buffer is empty and 1 means that the buffer is full. The parameters can be fine-tuned to manage the aggressiveness of the buffer model. The reason we use a concave function is that the client can more quickly switch to higher quality levels at the beginning, or recover from low buffer occupancy. Now we describe the semantics of the parameters α, b and c. The parameter α is set to the maximum available video rate, R_{max}. The parameter c enables a minimum buffer fill state, referred to as steady phase. This means that the client will download the segments with the lowest available video rate, until the buffer occupancy reaches the threshold B_{min}. Once the buffer occupancy increases above B_{min}, the algorithm enters the adaptation phase. The parameter c influences and modifies the range of the steady state. To set the value of B_{min} equal to 20% of the buffer size ($\delta = 0.2$), the value of c is set to $1/\delta = 5$. This means that until $\delta \leq 0.2$, the client stays in steady state, because for $c = 5$ and $\delta \leq 0.2$, the value of $\gamma \leq 0$. Setting the base of the logarithm function b equal to 5 would result in $\delta = 1$. To aggressively select the video rates, the video rate map selects the maximum available rate, R_{max}, when the buffer occupancy is 80% ($\delta = 0.8$) of the buffer size. Therefore, we use the base of 4 i.e. the parameter $b = 4$, which results in $\gamma = R_{max}$ (or a), when $\delta = 0.5$ and $c = 5$. As the value of δ increases, the client stays at the current video rate, so long as the value of γ suggested by (7) does not pass the value of the next highest available video rate.

When the concave function suggests a lower video rate, the client shifts to the linear function to select the video rates for the upcoming segments. The buffer model based

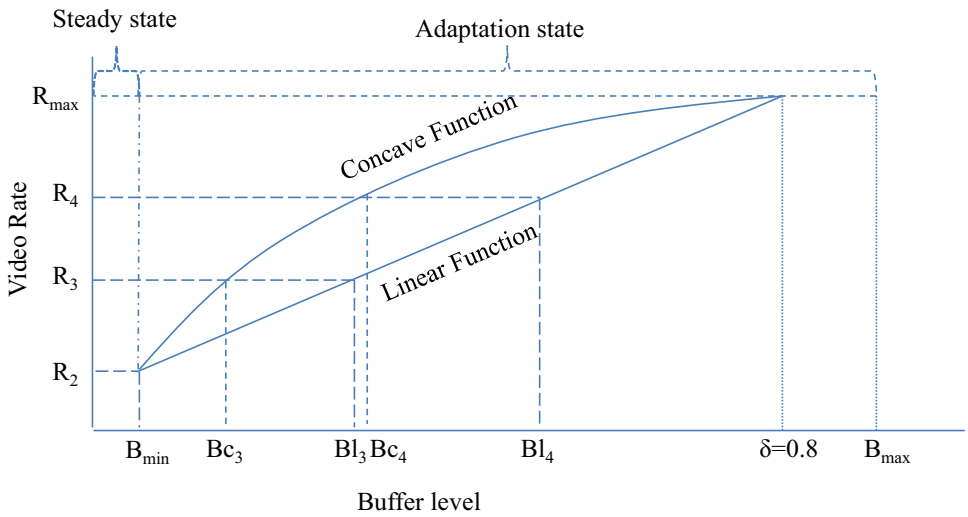


Fig. 7 The video rate map based on concave and linear buffer models

on the linear function to calculate the video rate restriction γ is given by:

$$\lambda(s_i) = \begin{cases} R_{\min} & B(i-1) < B_{\min} \\ R_{\min} + \dfrac{B(i-1) - B_{\min}}{0.8 \times B_{\max}} \times (R_{\max} - R_{\min}) & B_{\min} \leqslant B(i-1) < 0.8 \times B_{\max} \\ R_{\max} & B(i-1) \geqslant 0.8 \times B_{\max} \end{cases}. \quad (8)$$

As the buffer level drops, the client decides to switch to the lower video rates, since the depletion of the buffer indicates that the selected video rate is higher than the available throughput. One of the important objectives of the rate adaptation algorithms is to select the highest feasible video rate, but not at the expense of buffer underflow. The video clients do not have control over TCP sockets, and HTTP/1.1 does not support the termination of ongoing segment transfer, so the client can only switch to a different video rate when the segment download finishes. If the throughput suddenly drops in the middle of a segment transfer, the buffer may run dry before the client switches to a lower video rate. Authors in [24] suggest that the user experience improves when the video rate is increased aggressively as it makes the users believes that the provider is attempting to maximize the QoE. Figure 7 shows that the buffer threshold to select R_3 as suggested by concave function Bc_3 is greater than the buffer threshold suggested by Bl_3. This means that if the client adopts concave behaviour, it can more aggressively select the video rate, in comparison to if it adopts linear behaviour. When the estimated throughput and the buffer level drops, the client decides to switch to the less aggressive linear function, to avoid the risk of buffer underflow. Similar to the video rate map based on the concave function, the linear rate map always selects $R_{\min}$ when the buffer level drops below $B_{\min}$, and selects $R_{\max}$ when $\delta \geq 0.8$. As the buffer level decreases, the client stays at the current video rate so long as the value of γ suggested by (8) does not drop below the value of the next lowest available video rate. When the buffer level drops below $B_{\min}$, $R_{\min}$ is always selected.

First, we consider the scenario of an increase in throughput, and a subsequent increase in the buffer level. To increase the video rate in response to the increase in throughput and buffer level, four conditions should be satisfied:

1) $R\uparrow < T^E(i)$.
2) The buffer level should be greater than $Bc\uparrow$.
3) $R(i-1) \neq R_{\max}$.
4) $T^E(i) > T^E(i-1)$.

We denote video rates higher and lower than the current video rate by R↑ and R↓, respectively. To avoid buffer drop, the first condition makes sure that the selected video rate is less than the estimated throughput. As the video rate cannot be adapted until the download of the next segment, in the case of a sudden drop in throughput, the second condition reduces the probability of a buffer underflow event. As explained earlier, when the client decides to increase the video rate, it selects B_k using the concave function. The last condition reduces the frequency of video rate switches, by not switching up the video rate when the client observes a drop in throughput. This avoids the risk of a likely step down in the near future.

Next, we consider the scenario of a decrease in throughput, and a subsequent decrease in the buffer level. We stay at the current video rate, until the buffer level drops below Bc_k. This is to minimize the frequency of video rate switches, by not reacting to short-term fluctuations. Once the buffer level falls below Bc_k, we shift to the linear function to select the video rates. We continue to reduce the video rate, until the selected video rate is less than the estimated throughput and the buffer level is above the threshold Bl_k.

4.3 Smoothing video rate switches

The proposed algorithm selects the video rate based on both buffer occupancy and the estimated throughput. The variations in the playback buffer level results in video rate switches. Figure 8 shows examples of video rate switches due to variations in buffer level. In scenario 1, as the buffer level drops below the threshold Bc_k, the client steps down the video rate from video rate. In scenario 2, as the buffer level increases above Bc_k, the client steps up the video rate. A small fluctuation in throughput may result in fluctuation of the buffer level around Bc_k, which means a high frequency of video rate switches.

Many video streaming services encode their videos in variable bitrate (VBR). Encoding static scenes with fewer bits and active scene with more bits allows more flexibility and efficient utilization of bits. While all the segments are of equal duration, τ, the size of each segment varies. The larger segments will take more time to download, compared with the smaller segments. Therefore even in a stable network environment, as the client downloads segments of variable sizes, the buffer level may fluctuate. This results in higher video rate switches, which impair the viewing experience [2, 4].

To this end, we add a buffer zone around Bc_k, within which, if the buffer level stays, the client avoids switching the video rate. To explain how the proposed method smooths out the video rate switches, Fig. 9 introduces three scenarios. In scenario (a), the buffer level before downloading segment i lies between $\overline{B}$ and Bc_k. If the buffer level increases above Bc_k, the client switches up the video rate. In scenarios (b) and (c), the buffer level before downloading segment i is higher than Bc_k. If after downloading the ith segment, the buffer level stays between Bc_k and $\overline{B}$, the client does not switch the video rate; whereas, if the buffer level drops below $\overline{B}$, the client switches down the video rate. The buffer zone should be large enough to absorb the variations in buffer level, but not at the expense of risking buffer underflow. The larger the segment duration, the larger the expected variation in buffer level. Therefore, we set the value of $\overline{B}$to:

$$\overline{B} = Bc_k - \tau, \tag{9}$$

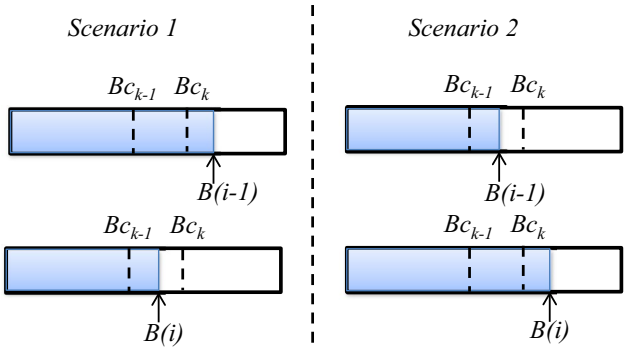


Fig. 8 An example of video rate switches due to variations in the buffer level

where τ is the segment duration.

5 Performance evaluation

5.1 Evaluation setup

We use network simulator, ns-3, as the experimental simulation environment. Our simulation adopts three rate adaptive algorithms as benchmarks. Besides the rate adaptive algorithm proposed in our previous work [7], we adopt the algorithms proposed in [12, 13, 19], as benchmarks to demonstrate the efficiency of the proposal. In the results, we refer to the algorithms proposed in [7, 12, 13, 19] as BBAB, AAA, SARA and MAL respectively. In the simulation, we evaluate the algorithm under varying network conditions, buffer sizes, and segment durations. We modified the code available in https://github.com/djvergad/dash to perform our experiments.

The length of the video is 400 s. To achieve adaptive streaming, the HTTP server offers the client seven levels of representation to adapt the video rates. These video rates are 356, 500, 800, 1200, 1800, 2500 and 3000 kbit/s. Figure 10 shows the topology implemented in this paper. The topology consists of an HTTP server, HTTP client, and a pair of routers. The link between the routers is our bottleneck link. To vary the throughput across the bottleneck, we add the UDP traffic between the routers.

5.2 Bitrate adaptation performance

First, we demonstrate how the proposed algorithm performs under multiple environments. For these experiments we set buffer size to 60 s. Figure 11 demonstrates the video rate selected by the proposed algorithm under a small throughput fluctuation scenario. This figure plots the values from the middle of the streaming session, as the objective is to show

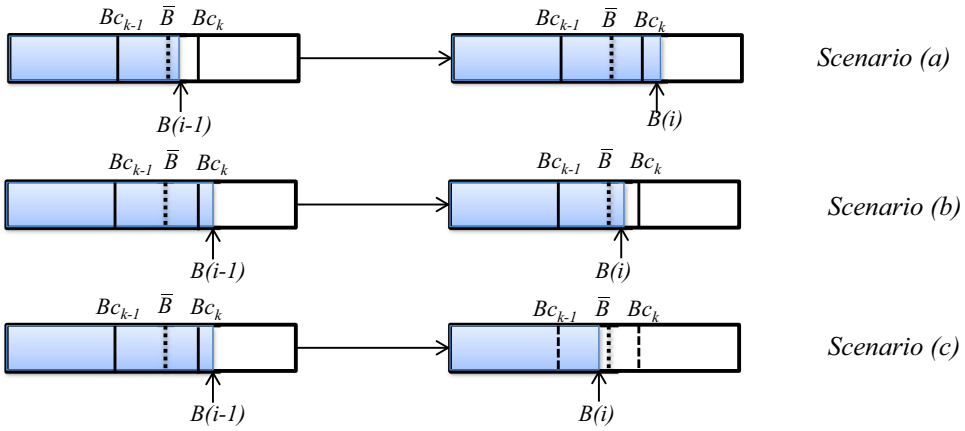


Fig. 9 Selection of video rates as the playback buffer level fluctuates

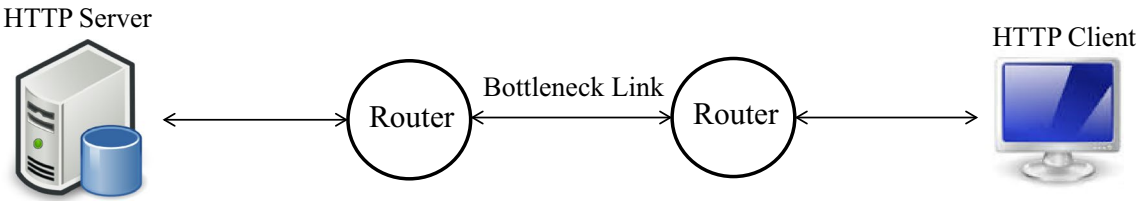


Fig. 10 The network topology

the response to short term fluctuations. Figure 11 shows that the proposed algorithm is stable in response to small fluctuations, which reduces the frequency of video rate switches. The reason for the stable response is that the buffer distance between $B_k(i)$ and $B_{k+1}(i)$ and the addition of the buffer zone provides a cushion, and reduces the frequency of video rate switches.

Figure 12 demonstrates the response of the proposed algorithm to a large throughput drop. An important property of an adaptive algorithm is that it should have a swift response to large fluctuations. To make sure that the throughput drop is not due to a short-term fluctuation, the proposed algorithm waits for the buffer level to drop below $\overline{B}$. Once the buffer level drops further, the algorithm quickly switches down the video rate, to avoid the risk of playback interruption. The proposed rate adaptive method quickly switches down the video rate, because in the case of a large variation in the throughput, the throughput estimation method proposed in Sect. 3 exponentially varies the throughput.

Figure 13a shows that as the throughput gradually increases, the proposed algorithm increases the video rate. As the algorithm adopts the concave behaviour when the buffer level increases, it quickly switches to higher video rates to efficiently utilize the throughput. When the throughput gradually drops, the proposed algorithm maintains a high video rate without risking buffer underflow. Figure 13b shows that the proposed algorithm ensures that a small drop

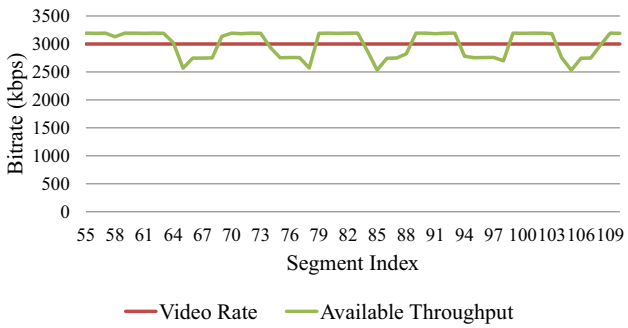


Fig. 11 The response of the proposed algorithm to small throughput fluctuations

in throughput doesn't result in unnecessary stepping down of the video rate.

5.3 Single-user scenario

The topology of a single-user scenario consists of an HTTP server, an HTTP client, and a pair of routers. We analyze the algorithms for the scenarios mentioned in Table 1. We demonstrate the impact of the buffer size and segment duration on the performance of the algorithms. The HTTP clients offer distinct buffer sizes. The rate adaptive algorithms should be able to guarantee QoE under different client settings. We set the buffer size to 20, 40 and 60 s, and evaluate the performance of the rate adaptive algorithms. Then,

we demonstrate how the algorithms perform as the segment duration varies. In the results, we refer to the throughput traces shown in Fig. 14a and b employed for single-user scenario as Scenario A and Scenario B respectively. Scenario A produces bandwidth fluctuations of variable amplitude. We evaluate how the algorithms adapt the video rate as the amplitude of throughput varies. Scenario B initially increases the video rate gradually and then produces large drops in the throughput of gradually increasing durations. We evaluate how the algorithms adjust the video rates when there are small and large variations in the throughput.

5.3.1 Scenario A

First, we set the buffer size and segment duration to 60 and 2 s respectively. Table 2 shows the statistics of the algorithms over the streaming session. The SARA algorithm results in the most fluctuating bitrate curve, and the AAA algorithm is the most stable, but to the detriment of video rate. The proposed method and the BBAB algorithms provide a smoother bitrate curve with higher bitrate. The proposed algorithm is able to achieve a high video rate when the network conditions improve. For downward switching, abrupt switching impairs the QoE, as compared to smooth switching [5, 25–27]. On average, the minimum quality level is rated 30% better quality in case of gradual video rate switches compared to an instantaneous switch [25]. The maximum downward switch when the client switches down the video rate means the largest video rate difference between any two consecutive segments over the whole session. Although the average of the switches and the standard deviation (STD) of the video rates selected by the proposed algorithm are higher, the lower value of the maximum downward switch shows that the proposed and AAA algorithms smoothly switch down the video rate. Unlike downward switching, viewers prefer abrupt increase in the video quality for upward switching [28, 29]. The higher value of the maximum upward switch shows that the proposed algorithm aggressively increases the video rate, to better utilize the available throughput. The MAL algorithm achieves high video rate and small number of video rate switches but experiences playback interruption for 1.6 s. The reason behind playback interruption is that the MAL algorithm does not check whether the estimated throughput is lower than the selected representation. Figure 15 shows the percentage of mid to high video rate segments downloaded by the client for Scenario A. As mentioned earlier that the user experience improves when the video rate is increased aggressively [24]. In addition, long spell of good quality video improves

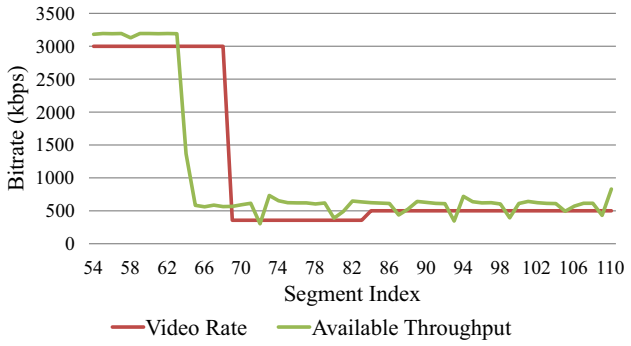
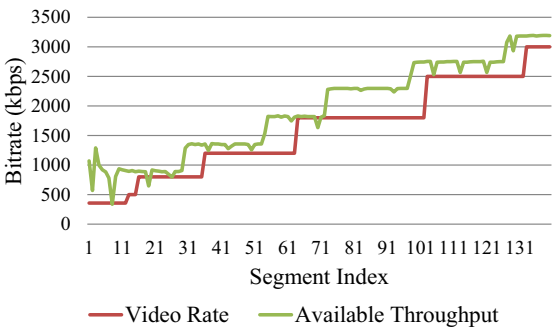


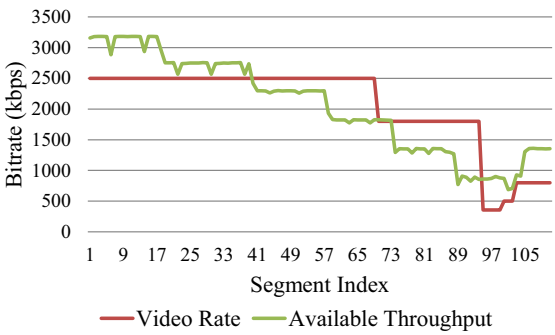
Fig. 12 The response of the proposed algorithm to a large throughput drop

Table 1 Buffer sizes and segment durations for the implemented scenarios

Scenario no.	Buffer size (s)	Segment duration (s)
1	60	2
2	40	2
3	20	2
4	60	4
5	60	10

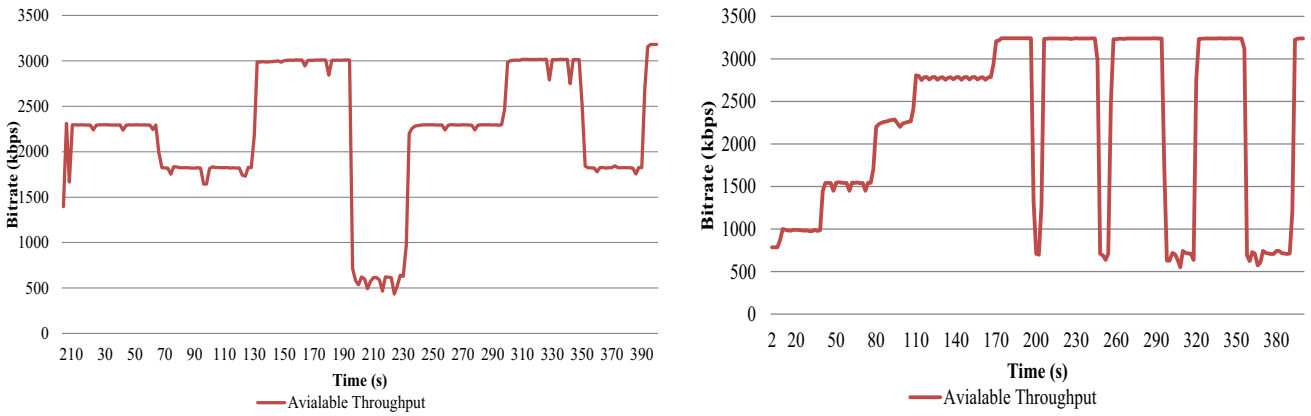


(a) A gradually increasing throughput scenario



(b) A gradually decreasing throughput scenario

Fig. 13 Response of the proposed algorithm to the gradually increasing throughput and decreasing throughput scenarios



(a) Throughput for scenario A

(b) Trace throughput for scenario B

Fig. 14 Throughput trace employed in the evaluation

the user experience [24, 26]. Figure 15 shows that only the proposed, SARA and MAL algorithms are able to efficiently utilize the bandwidth and download the segments encoded with the highest available video rates. However, the SARA and MAL algorithms stream the high quality video at the expense of high number of video rate switches and playback interruption respectively.

Table 3 shows the statistics of the algorithms when the buffer size is set to 40 s. The SARA algorithm achieves the highest average video rate, but results in the highest frequency of video rate switches. The SABA, BBAB, and MAL algorithms achieve high average video rate and low frequency of video rate switches. The AAA algorithm stabilizes the video rate curve, but achieves a low video rate. Table 3 shows that both the proposed and AAA algorithms smoothly switch down the video rate to improve the user experience. As explained earlier, the proposed algorithm has a higher average of switches, but this is due to an aggressive increase in the video rate in order to efficiently utilize the throughput. The MAL algorithm experiences playback interruption for 3.7 s. Similar to the previous experiment, the AAA and BBAB algorithms cannot stream the video at the highest available video rate.

Table 4 shows the statistics of the algorithms when the buffer size is set to 20 s. Similar to previous scenarios, the SARA algorithm achieves the highest average video rate, and results in the highest number of video rate switches. The BBAB algorithm achieves a video rate similar to the SABA algorithm when the buffer size is set to 60 and 40 s, but in the case of a smaller buffer size, the BBAB algorithm is nearly 400 kbps worse than the proposed method. The reason is that the BBAB algorithm requires large buffer sizes to select higher video rates. The AAA algorithm is the most stable method in the case of larger buffer sizes, but as the buffer size reduces, the frequency of the video rate switches increases. The proposed algorithm keeps the frequency of video rate switches low. The AAA algorithm conservatively increases the video quality, whereas the BBAB algorithm cannot select a video rate higher than 1800 kbps. SARA and SABA algorithms achieve high average video rate; but

Table 2 Statistics of different adaptive methods when buffer size is 60 s and segment duration is 2 s

Metrics/algorithms	Scenario A					Scenario B				
	Proposed	AAA	SARA	BBAB	MAL	Proposed	AAA	SARA	BBAB	MAL
Average video rate (kbps)	1903.29	1483.28	2169.14	1951.28	2131.00	1967.78	1281.23	2084.49	1901.35	2103.78
MAX (kbps)	3000	2500	3000	2500	3000	3000	2500	3000	2500	3000
MIN (kbps)	356	356	356	356	500	356	356	356	356	356
MAX Switch (kbps)	1444	1000	2644	1300	1700	1200	2000	2644	700	2200
Max switch downward (kbps)	1000	1000	2644	1300	1700	1200	2000	2644	700	2200
Max switch upward (kbps)	1444	1000	700	700	700	1000	1700	700	700	700
No. of switches	14	7	57	11	10	19	12	88	7	11
Avg of switches (kbps)	696	592	617.05	504	655.56	656.00	674.67	582.30	457.33	708.47
Standard deviation (kbps)	626.7	578.84	866.64	547.8	622.40	750.17	747.37	565.77	688.51	913.44

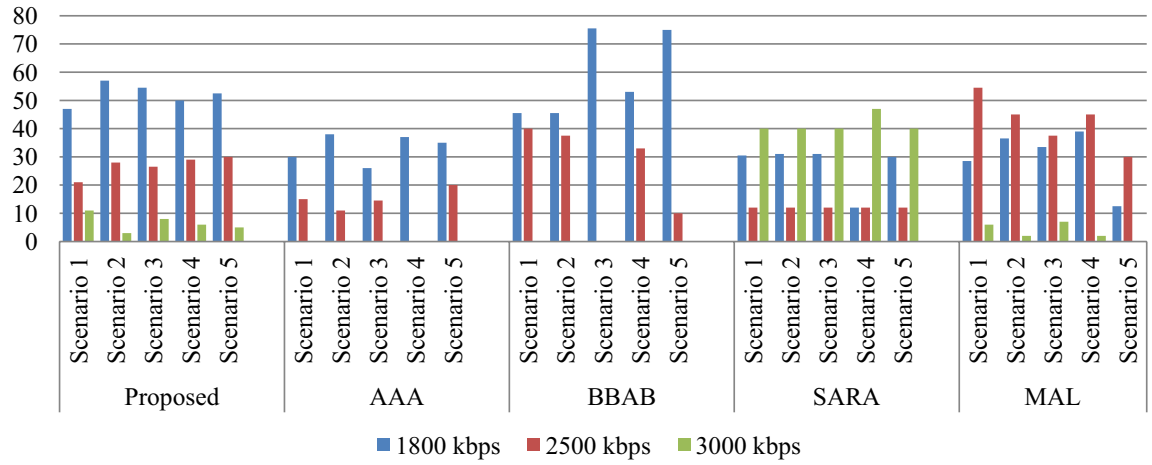


Fig. 15 The percentage of the time the client downloads the segments encoded at 1800, 2500 and 3000 kbps during the implementation of Scenario A

the SARA algorithm achieves it is at the expense of higher changes in the video rate. The MAL is the only algorithm that experiences playback interruption due to buffer underflow for 2.1 s.

Currently, video streaming services deploy segment duration differently in their services. Microsoft Smooth Streaming, Netflix and Apple HTTP Live Streaming offer segment duration of 2, 4 and 9 s respectively [30–32]. We set the segment durations to 2, 4 and 10 s, and evaluate the performance of the rate adaptive algorithms. The buffer size is set to 60 s for all of the experiments.

Table 5 shows the statistics of the algorithms results when the segment duration is set to 4 s. In comparison to the experiment where the segment size is set to 2 s, all of the algorithms except the AAA algorithm achieve similar average video rates. The average video rate of the AAA algorithm drops by roughly 200 kbps. The AAA algorithm has the most stable video rate curve followed by the MAL algorithm, whereas the SARA algorithm results in the highest frequency of video rate switches. Table 5 shows that the SABA algorithm has a slightly higher number of video rate switches as compared to the BBAB algorithm, but switches down the video rate more smoothly. In case of the SABA algorithm, the majority of video rate switches are between high and mid-quality. The experiments have shown that limited noticeable difference in quality is observed between high and mid-quality switches during video playback [25]. The AAA algorithm downloads the highest percentage of low quality segments.

In the next experiment, we increase the segment duration to 10 s. Table 6 shows that the BBAB algorithm has a stable response, but the average video rate drops. Furthermore, BBAB algorithm does not experience a downward switch. Figure 15 shows that the BBAB algorithm streams more than 70% of the video at 1800 kbps. It means that the algorithm does not efficiently utilize the bandwidth. The SABA algorithm outperforms BBAB by almost 227 kbps. Also, it smoothly changes the video rate to minimize the degradation

Table 3 Statistics of different adaptive methods when buffer size is 40 s and segment duration is 2 s

Metrics/algorithms	Scenario A					Scenario B				
	Proposed	AAA	SARA	BBAB	MAL	Proposed	AAA	SARA	BBAB	MAL
Average video rate (kbps)	1901.78	1460.82	2169.14	1922.67	1944.64	1964.33	1343.34	2089.28	1901.38	2103.28
MAX (kbps)	3000	2500	3000	2500	300	3000	2500	3000	2500	3000
MIN (kbps)	356	356	356	356	356	356	356	356	356	500
MAX Switch (kbps)	1444	1000	2644	1300	2000	1300	2000	2644	1000	2200
Max switch downward (kbps)	700	700	2644	1300	2000	1300	2000	700	1000	2200
Max switch upward (kbps)	1444	1000	700	700	700	1000	1300	2644	700	700
No. of switches	13	11	59	12	13	22	18	88	9	13
Avg of switches (kbps)	711.07	521.09	596.13	512	565.67	648.00	686.00	575.68	527.11	768
Standard deviation (kbps)	581.26	590.37	866.64	557.76	707.58	879.40	746.07	1005.67	781.94	939

Table 4 Statistics of different adaptive methods when buffer size is 20 s and segment duration is 2 s

Metrics/algorithms	Experiment A					Experiment B				
	Proposed	AAA	SARA	BBAB	MAL	Proposed	AAA	SARA	BBAB	MAL
Average video rate (kbps)	1948.29	1437.19	2169.14	1558.78	1913	1937.66	1288.19	2088.97	1774.766	1968.61
MAX (kbps)	3000	2500	3000	1800	3000	3000	2500	3000	2500	3000
MIN (kbps)	356	356	356	356	500	356	356	356	356	356
MAX switch (kbps)	1444	700	2644	600	2000	1200	2144	2644	1700	2200
Max switch downward (kbps)	1000	700	2644	600	2000	1000	2144	2644	1700	2200
Max switch upward (kbps)	1444	700	700	600	700	1200	1700	1200	700	700
No. of switches	12	16	59	10	11	29	19	87	22	17
Avg of switches (kbps)	770.33	514.5	596.13	404.4	604	624.82	933.26	582.30	702.10	721.74
Standard deviation (kbps)	626.48	618.33	866.64	456.87	748.41	911.51	754.83	1016.36	760.86	959.46

of the user experience, and has a low average video rate switch. Like the previous experiments, the SARA algorithm achieves a higher average video rate, and experiences higher video rate switches, while MAL algorithm achieves the lowest average video rate. Figure 15 shows that only the proposed and SARA algorithms are able to stream the video at the highest available video rate. The SARA algorithm streams the video at the highest available video rate at the expense of high number of video rate switches and playback interruptions. The SARA and MAL algorithms experience playback interruptions for 20.8 and 0.8 s, respectively. The SABA algorithm downloads a high percentage of high quality segments, and smoothly switches the video rate.

5.3.2 Scenario B

In this section we evaluate the algorithms for Scenario B. Table 2 shows that the SARA and MAL algorithms achieve high video rate but at the expense of higher video rate changes and playback interruptions, respectively. The MAL algorithm experiences playback interruptions six times during the streaming session. The proposed algorithm achieves high average video rate. It experiences a slightly higher number of video rate switches because the proposed algorithm aggressively reduces the video rate when the throughput suddenly drops to mitigate the risk of buffer underflow. The BBAB algorithm has a lower average video rate and lower number of video rate switches than the proposed algorithm. Figure 16 shows that similar to Scenario A, only the proposed, SARA and MAL algorithms stream the video at the highest available video rate. The SARA and AAA algorithms download high percentage of video rates encoded at 3000 kbps, but to the detriment of the user experience as they result in high number of video rates switches and playback interruption, respectively. Figure 17 shows that the BBAB algorithm achieves the highest eMOS values followed by the proposed algorithm. The eMOS of the SABA algorithm is 0.04 worse than the BBAB algorithm.

Then we evaluate the algorithms for the scenario when the buffer size is set to 40 s. The SARA algorithm results in the highest number of video rate of switches, whereas the MAL algorithm results in three playback interruptions, which degrade the user's experience. The BBAB algorithm has slightly less number of video rate switches because it takes a conservative approach in selecting higher video rates; therefore, it never selects the highest available video rate. The BBAB algorithm is nearly 60 kbps worse than the proposed method. Similar to the previous scenario, the BBAB and AAA algorithms cannot stream the video at the highest available video rate. The AAA algorithm achieves the lowest video rate among the compared algorithms. The SABA algorithm achieves the highest eMOS value followed by the BBAB algorithm.

Table 5 Statistics of different adaptive methods when buffer size is 60 s and segment duration is 4 s

Metrics/Algorithms	Scenario A					Scenario B				
	Proposed	AAA	SARA	BBAB	MAL	Proposed	AAA	SARA	BBAB	MAL
Average video rate (kbps)	1934.92	1280.96	2161.92	1927.56	2020.00	1962.56	1240.65	2096.65	1807.71	2048.94
MAX (kbps)	3000	1800	3000	2500	3000	300	2500	3000	2500	3000
MIN (kbps)	356	356	356	356	800	356	356	356	356	356
MAX switch (kbps)	1444	1444	2644	2500	700	700	2000	2644	1000	2200
Max switch downward (kbps)	1444	1444	2644	1700	700	700	2000	2644	1000	2200
Max switch upward (kbps)	844	844	700	700	700	700	2000	700	700	700
No. of switches	15	6	51	8	7	17	11	44	7	9
Avg of switches (kbps)	655.46	722.00	694.03	693.00	600.00	585.78	802.67	675.18	549.14	649.33
Standard deviation (kbps)	626.70	499.01	982.17	502.77	576.83	692.24	758.59	1001.40	732.22	960.17

Table 6 Statistics of different adaptive methods when buffer size is 60 s and segment duration is 10 s

Metrics/algorithms	Scenario A					Scenario B				
	Proposed	AAA	SARA	BBAB	MAL	Proposed	AAA	SARA	BBAB	MAL
Average video rate (kbps)	2018.9	1540.3	2310.9	1791.61	1497.5	2003.80	1335.3	1963.1	1521.4	1974.53
MAX (kbps)	3000	2500	3000	2500	2500	3000	2500	3000	2500	3000
MIN (kbps)	356	356	356	356	800	356	356	356	356	356
MAX switch (kbps)	1444	1000	2644	2500	1700	700	1700	2644	1300	700
Max switch downward (kbps)	700	1000	2644	1700	1700	700	1700	2500	1300	700
Max switch upward (kbps)	1444	1000	2500	844	700	700	1700	2644	700	700
No. of switches	10	5	14	3	7	11	8	22	10	12
Avg of switches (kbps)	714.40	828.80	1493.43	714.67	728.57	570.33	1027.11	1041.82	614.40	570.33
Standard deviation (kbps)	534.99	643.48	1114.00	381.66	775.09	709.51	750.77	1059.59	717.02	709.51

In the next experiment we set the buffer size to 20 s. Like the previous experiments, SARA and MAL algorithms result in high number of video rate switches and playback interruptions, respectively. The MAL algorithm experiences nine playback interruptions, which degrade the user's experience. The proposed algorithm achieves a high average video rate. It experiences slightly high number of video rate switches. The reason is that due to small buffer size and the ability of the proposed algorithm to efficiently utilize the bandwidth, when the throughput increases the proposed algorithm quickly increases the video rate and as the throughput drops, it decreases the video rate, while making sure that it does not experience buffer underflow. Figure 16 shows that the proposed algorithm streams the majority of the video at high video rates. The SABA algorithm achieves the highest eMOS value followed by the BBAB algorithm.

Next, we set the buffer size to 60 s and increase the segment duration to 4 s. The proposed algorithm selects high video rate while avoiding the playback interruption. The MAL algorithm experiences three playback interruptions due to buffer underflow. The BBAB algorithm results in the lowest number of video rate changes but it achieves average video rate 150 kbps worse than the proposed algorithm. Figure 17 shows that the proposed algorithm achieves the best eMOS among the compared algorithms.

Next, we increase the segment duration to 10 s. The proposed algorithm achieves the highest average video rate followed by MAL. The MAL algorithm experiences four playback interruptions. The AAA algorithm results in the lowest average video rate, whereas the SARA algorithm experiences the highest number of video rate changes. Similar to Scenario A, in case of BBAB algorithm, the average video rate drops significantly. Figure 16 shows that only the proposed and SARA algorithm is able to select the segments encoded at the highest available video rate. The AAA and BBAB algorithms stream majority of the video at low video rates. Figure 17 shows that the eMOS value of the BBAB algorithm drops sharply as the segment duration is increased to 10 s. The proposed algorithm achieves the highest eMOS value.

The experiments show that the BBAB algorithm keeps the frequency of video rate switches low, but when the buffer size drops to 20 s or the segment duration is increased to 10 s, the average video rate drops significantly. The SARA algorithm achieves a high average video rate, but at a high frequency of video rate switches. On the other hand, the

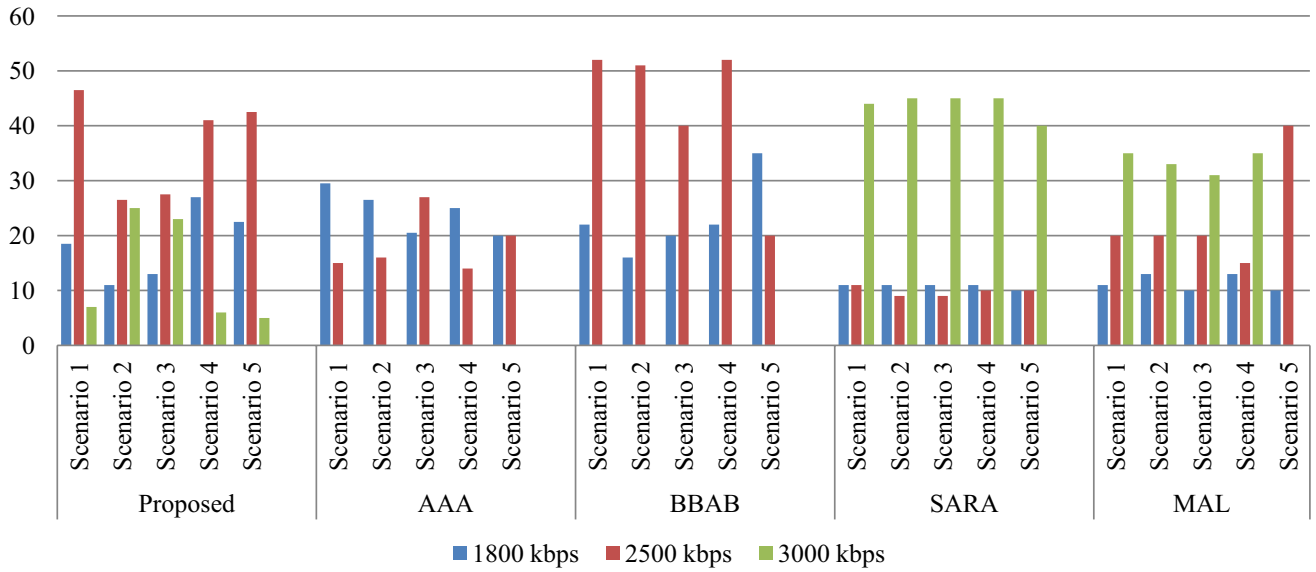


Fig. 16 The percentage of the time the client downloads the segments encoded at 1800, 2500 and 3000 kbps during the implementation of Scenario B

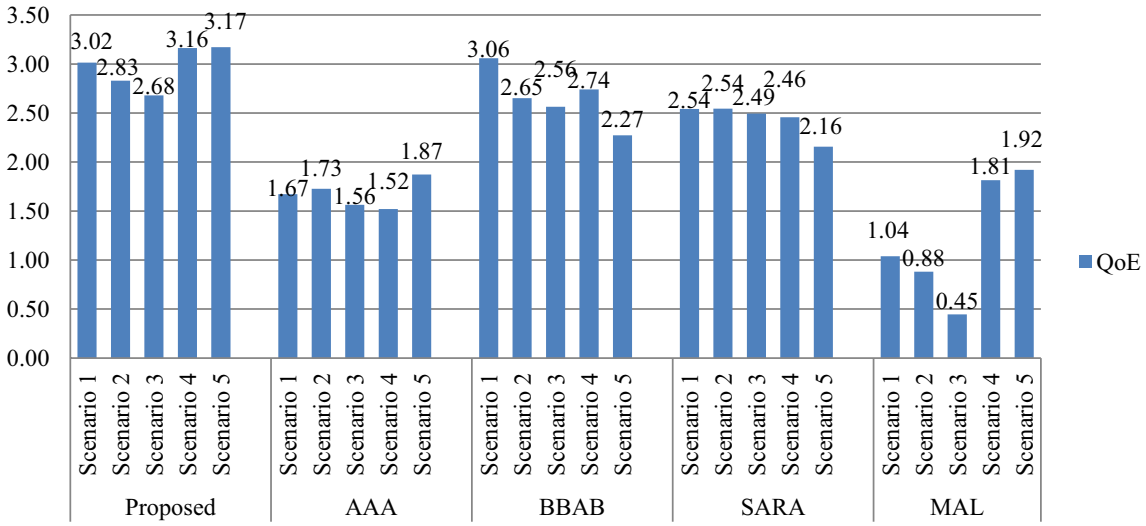


Fig. 17 eMOS values of the algorithms for scenario B

AAA algorithm stabilizes the video rate curve, but achieves the lowest video rate among all the algorithms. The MAL algorithm stabilizes video rate changes but at the expense of playback interruptions, which degrades the user's experience. The proposed algorithm irrespective of the buffer size and segment duration achieves high video rate and minimizes video rate switches while avoiding playback interruption. Furthermore, assures a smooth switch from the higher rate to the lower video rate.

5.4 Multi-client scenario

In this section, we analyze the performance of the algorithms when multiple clients share the bottleneck. Figure 18 shows the topology implemented for the multi-user scenario. The bandwidth of the bottleneck link is 10 mbps for all experiments. The algorithms are evaluated for varying number of clients, buffer sizes, and segment durations. Similar to the single client-scenario, we set the buffer size to 20, 40 and 60 s and set the segment duration to 2, 4 and 10 s.

In an environment where multiple clients compete for the bottleneck, the clients are inefficient and select low-quality video rates and bandwidth is shared unfairly among the competing clients. The *inefficiency* at time t is given by the following [15]:

$$\text{Inefficiency} = \frac{\left|\sum_x b_{x,t} - W\right|}{W}, \tag{10}$$

where W is the bandwidth, and each client x selects bit rate $b_{x,t}$. [33] defines the *unfairness* metric for two competing clients as the average of the absolute bit rate differences between the corresponding segments requested by each client. To evaluate *unfairness*, this is generalized to multiple players as $\sqrt{1 - JainFair}$, where *JainFair* is the Jain fairness index [34] of $b_{x,t}$ over all players. Let $R^A = \{R^A_1, R^A_2, \ldots, R^A_n\}$ denote the set that contains the video rates achieved by n competing clients. Additionally, to evaluate the unfairness metric for more than two users, we define the parameter *diff* as follows:

$$diff = Max\left(R^A\right) - Min\left(R^A\right). \tag{11}$$

Ideally, the values of *inefficiency, unfairness* and *diff* should be zero. Low values of *inefficiency, unfairness* and *diff* are desired; a low value of *inefficiency* means that the client selects the highest feasible bit rates lower than the actual throughput and low values of *diff and unfairness* means that the competing clients achieve equitable video rates.

5.4.1 Five-clients scenario

In this section we compare the adaptive algorithms when five clients share the bottleneck. We evaluate the algorithms for scenarios mentioned in Table 1. The results show that the proposed scheme provides the best performance among the compared algorithms overall. The proposed algorithm results in low *inefficiency, unfairness* and *diff* values. Figure 19 shows that the SARA and SABA algorithm are the least inefficient whereas the AAA algorithm is the most inefficient algorithm. Figure 20 shows the *diff* values of the compared algorithms for each scenario. Figure 21 shows the average *diff* value of the compared algorithms. Figures 19 and 20 show that the proposed algorithm has the lowest *unfairness* and *diff* values. Figures 21 and 22 show that the proposed algorithm makes sure that the client switches the video rate gradually while keeping the video rate switches low. The performance of the BBAB algorithm fluctuates from one environment to another. The BBAB algorithm shows improved performance for large buffer sizes and small segment durations, but when the segment duration is increased or buffer size is decreased, the performance of the BBAB algorithm degrades. The SARA algorithm has a low *inefficiency* value but results in a high *unfairness* and *diff* values. Furthermore, it experiences most number of video rate changes. The AAA algorithm has a high *inefficiency, unfairness* and *diff* values. Figure 22 shows the video rate switches experienced by the algorithms. The BBAB algorithm has the

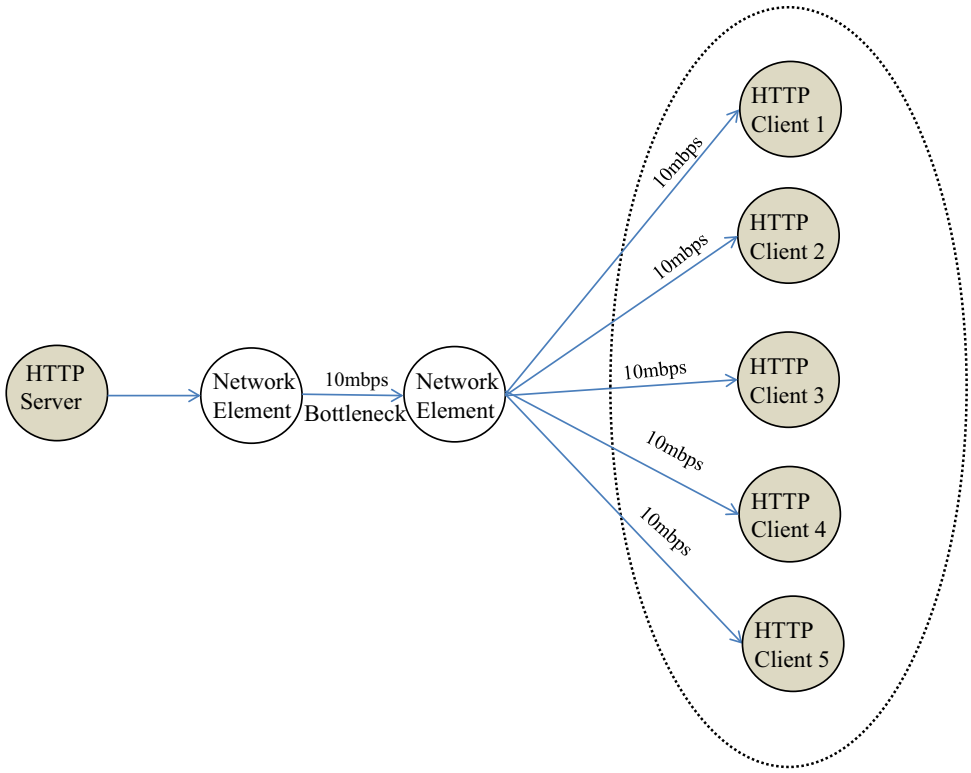


Fig. 18 Multi-client network topology

lowest number of video rate switches. As explained earlier, a large downward video rate switch affects user's experience more than a gradual downward switch. The proposed algorithm on average experiences 1.25 more video rate switches per 100 s compared to BBAB algorithm but the proposed algorithm results in lower average downward switch. The MAL algorithm performs the worst among the compared algorithms as it experiences a high number of video rate switches. The clients experience on average 10 playback interruptions during the implementation of the MAL algorithm for each scenario. The rest of the algorithms streamed the video without experiencing playback interruption.

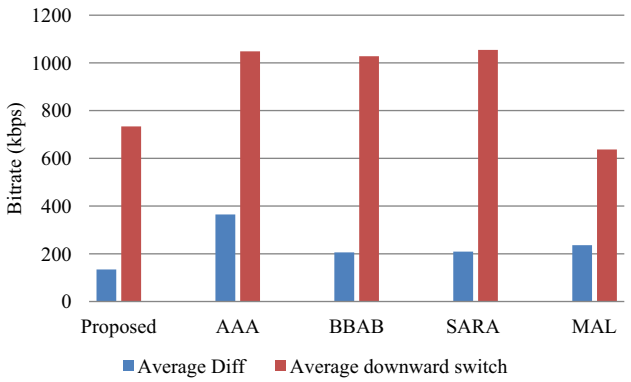
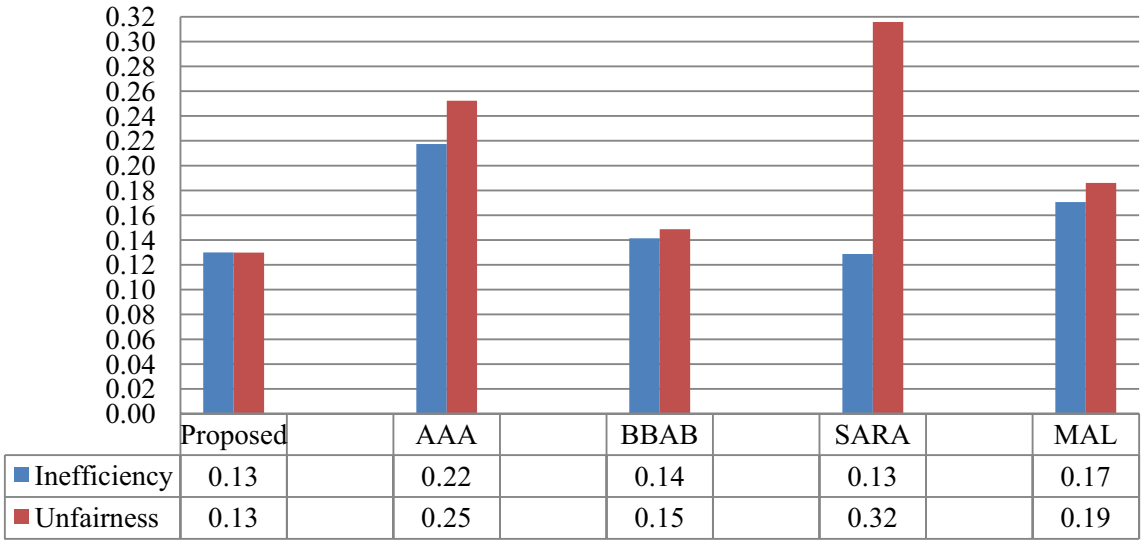


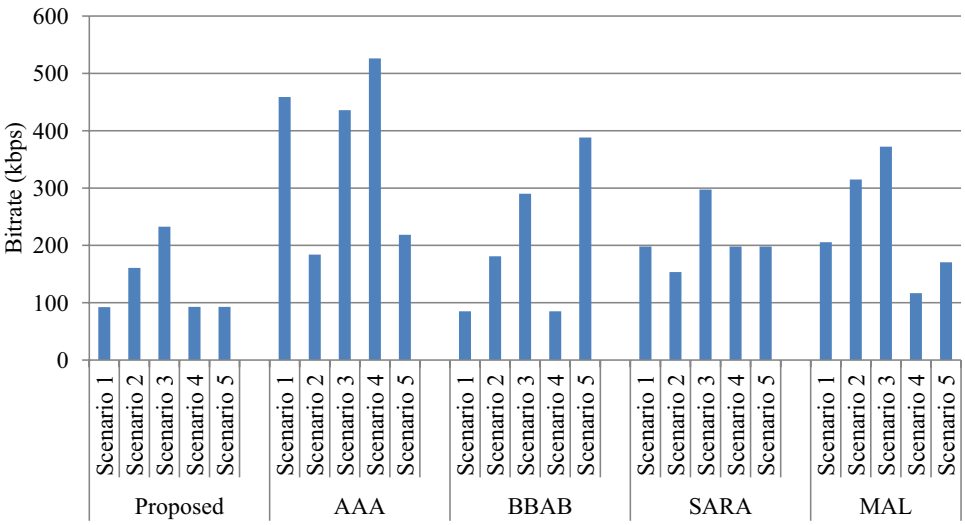
Fig. 21 Average *diff* and downward switch values of the compared algorithms



	Proposed	AAA	BBAB	SARA	MAL
Inefficiency	0.13	0.22	0.14	0.13	0.17
Unfairness	0.13	0.25	0.15	0.32	0.19

Fig. 19 *Inefficiency* and *unfairness* of the compared algorithms

Fig. 20 *diff* values of the compared algorithms for each scenario mentioned in Table 1



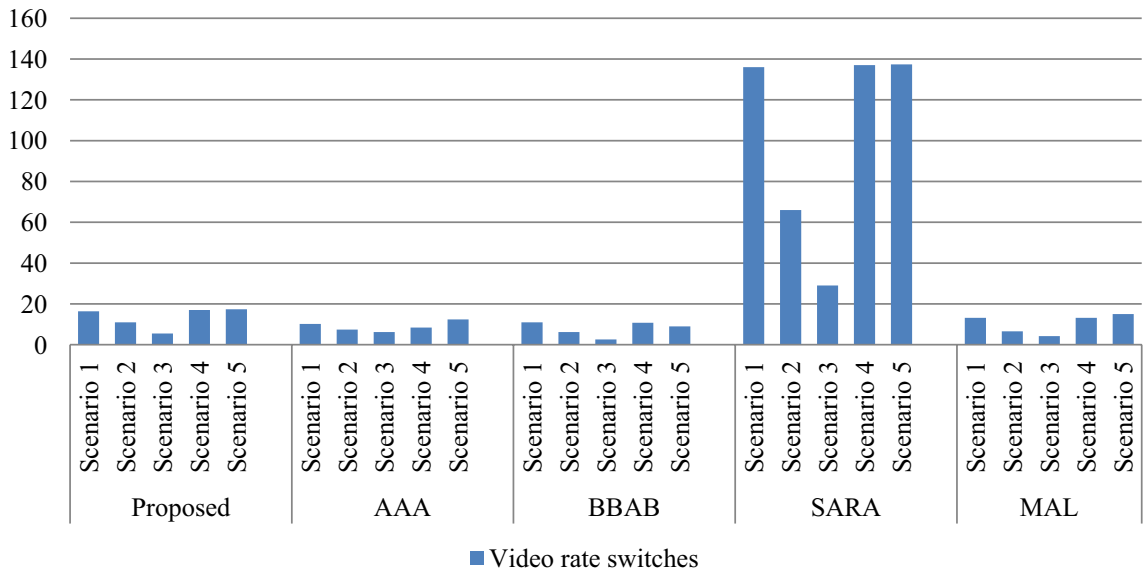


Fig. 22 Video rate switches for each scenario mentioned in Table 1

6 Conclusion

In this paper, we propose a throughput estimation method and a rate adaptive algorithm for HTTP adaptive streaming. The proposed throughput estimation method can accurately estimate the throughput of the upcoming segment based on previous samples. We show that the throughput estimation method is sensitive to large fluctuations, and robust to small fluctuations. The objective of the adaptive bitrate streaming algorithm is to improve the viewing experience of multimedia streaming applications. The proposed rate adaptive algorithm offers stable response during small variations in the throughput to minimize the video rate switches, and quickly reacts when there are large variations in throughput to minimize the risk of buffer underflow. The proposed algorithm provides high quality video by achieving a high video rate while preventing interruption in playback. We show that the algorithm minimizes the video rate switches, and makes sure that the video rate changes smoothly to improve the user experience. Furthermore, we show that in a multi-client scenario, the proposed algorithm efficiently utilizes the bandwidth and the competing clients achieve comparable video rates.

Acknowledgements This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2017-0-00224, Development of generation, distribution and consumption technologies of dynamic media based on UHD broadcasting contents). It has also been conducted by the Research Grant of Kwangwoon University in 2018.

References

1. Dobrian, F., Awan, A., Joseph, D., Ganjam, A., Zhan, J., Sekar, V., Stoica, I., Zhang, H.: Understanding the impact of video quality on user engagement. ACM SIGCOM Comput. Commun. Review. **56**(3), 91–99 (2013). https://doi.org/10.1145/2018436.2018478
2. Ni, P., Eg, R., Eichhorn, A., Griwodz, C., Halvorsen, P.: Flicker effects in adaptive video streaming to handheld devices. In: Proceedings of ACM Int. Conf. on Multimedia, pp. 463–472. Arizona, USA (2011). https://doi.org/10.1145/2072298.2072359
3. Liu, Y., Dey, S., Gillies, D., Ulupinar, F., Luby, M.: User experience modeling for DASH video. In: Proceedings of IEEE Packet Video Workshop, pp. 1–8. San Jose, USA, (2013). https://doi.org/10.1109/pv.2013.6691459
4. Shen, Y.., Yitong, L., Yang, H., Yang, D.: Quality of Experience study on dynamic adaptive streaming based on HTTP. IEICE Tran. Commun. **98**(1), 62–70 (2015). https://doi.org/10.1587/transcom.e98.b.62
5. Egger, S., Gardlo, B., Seufert, M., Schatz, R.: The impact of adaptation strategies on perceived quality of http adaptive streaming. In: Proceedings of ACM Workshop Design, Quality and Deployment Adaptive Video Streaming, pp. 31–36. Sydney, Australia: (2014). https://doi.org/10.1145/2676652.2676658
6. Dubin, R., Hadar, O., Dvir, A.: The effect of client buffer and MBR consideration on DASH adaptation logic. In: Proceedings of IEEE Wireless Commun. and Networking Conference, pp. 2178–2183. Shanghai, China: (2013). https://doi.org/10.1109/wcnc.2013.6554900
7. Rahman, W., Chung, K.: Buffer-based adaptive bitrate algorithm for streaming over HTTP. KSII Tran. Internet and Inform. Syst. **9**(11), 4585–4622 (2015). https://doi.org/10.3837/tiis.2015.11.019
8. VideoLAN. (2013). Vlc sourece code. [Online]. Available: http://www.videolan.org/ vlc/download-sources.html
9. Akhshabi, S., Begen, A.C., Dovrolis, C.: An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In: Proceedings of ACM Conf. on Multimedia System, pp. 157–168. California, USA: (2011). https://doi.org/10.1145/1943552.1943574

10. Thang, T.C., Ho, Q.D., Kang, J.W., Pham, A.T.: Adaptive streaming of audiovisual content using MPEG DASH. IEEE Trans. Consumer Electron. **58**(1), 78–85 (2012). https://doi.org/10.1109/tce.2012.6170058
11. Liu, C., Bouazizi, I., Gabbouj, M.: Rate adaptation for adaptive HTTP streaming. In: Proceedings of ACM Conf. on Multimedia Syst., pp. 169–174. California, USA: (2011). https://doi.org/10.1145/1943552.1943575
12. Miller, K., Quacchio, E., Gennari, G., Wolisz, A.: Adaptation algorithm for adaptive streaming over HTTP. In: Proceeding of IEEE Packet Video Workshop, pp. 173–178. Munich, Germany: (2012). https://doi.org/10.1109/pv.2012.6229732
13. Juluri, P., Tamarapalli, V., Medhi, D.: SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP. In: Proceedings of IEEE Int. Conf. on Commun. Workshop, pp. 1765–1770. London, United Kingdom: (2015). https://doi.org/10.1109/iccw.2015.7247436
14. Le, H.T., Nguyen, D.V., Ngoc, N.P., Pham, A.T., Thang, T.C.: Buffer-based bitrate adaptation for adaptive HTTP streaming. In: Proceedings of IEEE Conf. on Advanced Technol. Commun., pp. 33–38. Hochiminh, Vietnam: (2013). https://doi.org/10.1109/atc.2013.6698072
15. Jiang, J., Sekar, V., Zhang, H.: Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In: Proceedings of ACM Int. Conf. on Emerging Networking Experiments and Technol., pp. 97–108. Nice, France: (2012). https://doi.org/10.1145/2413176.2413189
16. Huang, T.Y., Johari, R., McKeown, N., Trunnell, M., Watson, M.: A buffer-based approach to rate adaptation: evidence from a large video streaming service. ACM SIGCOM Comput. Commun. Review. **44**(4), 187–198 (2015). https://doi.org/10.1145/2619239.2626296
17. Rahman, W., Chung, K.: Chunk size aware buffer-based algorithm to improve viewing experience in dynamic HTTP streaming. IEICE Tran. Commun. **E99-B**(3), 767–775 (2016). https://doi.org/10.1587/transcom.2015ebp3398
18. Sieber, C., Hossfeld, T., Zinner, T., Tran-Gia, P., Timmerer, C.: Implementation and user-centric comparison of a novel adaptation logic for dash with SVC. In: Proceedings of IEEE Int. Symposium on Integrated Network Management, pp. 1318–1323. Ghent, Belgium: (2013). https://doi.org/10.1109/inm.2005.1440752
19. Dubin, R., Dvir, A., Hadar, O., Harel, N., Barkan, R.: Multicast adaptive logic for dynamic adaptive streaming over http network. In: Proceedings of IEEE Conf. on Comput. Commun. Workshops, pp. 269–274. Hong Kong: (2015). https://doi.org/10.1109/infcomw.2015.7179396
20. Dubin, R., Dvir, A., Pele, O., Hadar, O., Katz, I., Mashiach, O.: Adaptation logic for HTTP dynamic adaptive streaming using geo-predictive crowdsourcing for mobile users. Multimedia Syst. (2016). https://doi.org/10.1007/s00530-016-0525-6
21. Huang, T.Y., Nikhil, H., Brandon, H., Nick, M., Ramesh, J.: Confused, timid, and unstable: picking a video streaming rate is hard. In: Proceedings of ACM Int. Conf. on Internet Measurement, pp. 225–238. Boston, USA: (2012). https://doi.org/10.1145/2398776.2398800
22. Egger, S., Hossfeld, T., Schatz, R., Fiedler, M.: Waiting times in quality of experience for web based services. In: Proceedings of IEEE Int. Workshop on Quality Multimedia Experience, pp. 86–96. Melbourne, Australia: (2012). https://doi.org/10.1109/qomex.2012.6263888
23. Mueller, C., Stefan, L., Grandl, R., Timmerer, C.: Oscillation compensating dynamic adaptive streaming over HTTP. In: Proceedings of IEEE Int. Conf. on Multimedia and Expo, pp. 1–6. Torino, Italy: (2015). https://doi.org/10.1109/icme.2015.7177435
24. Moorthy, K., Choi, L.K., Bovik, A.C., De Veciana, G.: Video quality assessment on mobile devices: subjective, behavioral and objective studies. IEEE J. Sel. Topics Signal Process. **6**(6), 652–671 (2012). https://doi.org/10.1109/jstsp.2012.2212417
25. Staelens, N., De Meulenaere, J., Claeys, M., Van Wallendael, G., Van den Broeck, W., De Cock, J., Van de Walle, R., Demeester, P., De Turck, F.: Subjective quality assessment of longer duration video sequences delivered over HTTP adaptive streaming to tablet devices. IEEE Trans. Broadcast. **60**(4), 707–714 (2014). https://doi.org/10.1109/tbc.2014.2359255
26. Hoßfeld, T., Seufert, M., Sieber, C., Zinner, T.: Assessing effect sizes of influence factors towards a QoE model for HTTP adaptive streaming. In: Proceedings of IEEE Int. Conf. on Multimedia and Expo, pp. 111–116. Singapore: (2014). https://doi.org/10.1109/qomex.2014.6982305
27. Zink, M., Jens, S., Ralf, S.: Layer-encoded video in scalable adaptive streaming. IEEE Trans. Multimedia **7**(1), 75–84 (2005). https://doi.org/10.1109/TMM.2004.840595
28. Zink, M., Künzel, O., Schmitt, J., Steinmetz, R.: Subjective Impression of Variations in Layer Encoded Videos. In: Proceeding of Int. Workshop on Quality of Service, pp. 137–154. Berkeley, CA, USA: (2003). https://doi.org/10.1007/3-540-44884-5_8
29. Grafl, M., Timmerer, C.: Representation switch smoothing for adaptive HTTP streaming. In: Proceedings of the Int. Workshop on Perceptual Quality of Systems, pp. 178–183. Vienna, Austria: (2013). https://doi.org/10.21437/PQS.2013-32
30. Zambelli, A.: Microsoft Corporation. IIS smooth streaming technical overview. [Online]. Available: https://docs.microsoft.com/en-us/iis/media/on-demand-smooth-streaming/smooth-streaming-technicaloverview. Accessed 10 Feb 2018
31. Adobe. Configure HTTP Dynamic Streaming and HTTP Live Streaming. [Online]. Available: https://helpx.adobe.com/adobe-media-server/dev/configure-dynamic-streaming-live-streaming.html. Accessed 10 Feb 2018
32. Pantos, R.: HTTP Live Streaming. [Online]. https://tools.ietf.org/html/rfc8216. Accessed 10 Feb 2018
33. Akhshabi, S., Anantakrishnan, L., Begen, A.C., Dovrolis, C.: What happens when HTTP adaptive streaming players compete for bandwidth?. In: Proceedings of ACM Workshop Netw. and Operating Syst. Support for Digital Audio and Video, pp. 9–14. Toronto, Canada: (2012). https://doi.org/10.1145/2229087.2229092
34. Jain, R., Chiu, D., Hawe, W.: A quantitative measure of fairness and discrimination for resource allocation in shared computer system. Technical Report, DEC, 1984