Buffer-Based Adaptive Bitrate Algorithm for Streaming over HTTP

Waqas ur Rahman and Kwangsue Chung
Department of Electronics and Communications Engineering, Kwangwoon University
447-1, Wolgye-dong, Nowon-gu, Seoul, Korea
[e-mail: waqas@cclab.kw.ac.kr, kchung@kw.ac.kr]
*Corresponding author: Kwangsue Chung

Received May 6, 2015; revised August 8, 2015; accepted September 11, 2015; published November 30, 2015

Abstract

Video streaming services make up a large proportion of Internet traffic on both fixed and mobile access throughout the world. Adaptive streaming allows for dynamical adaptation of the bitrate with varying network conditions, to guarantee the best user experience. Adaptive bitrate algorithms face a significant challenge in correctly estimating the throughput as it varies widely over time. In this paper, we first evaluate the throughput estimation techniques and show that the method that we have used offers stable response to throughput fluctuations while maintaining a stable playback buffer. Then, we propose an adaptive bitrate scheme that intelligently selects the video bitrates based on the estimated throughput and buffer occupancy. We show that the proposed scheme improves viewing experience by achieving a high video rate without taking unnecessary risks and by minimizing the frequency of changes in the video quality. Furthermore, we show that it offers a stable response to short-term fluctuations and responds swiftly to large fluctuations. We evaluate our algorithm for both constant bitrate (CBR) and variable bitrate (VBR) video content by taking into account the segment sizes and show that it significantly improves the quality of video streaming.

Keywords: Video Streaming, Quality Adaptation, Rate Adaptation, Quality of Experience

This work was supported by the Institute for Information & Communications Technology Promotion(IITP) grant funded by the Korea government (MSIP) (No.R0101-15-293, Development of Object-based Knowledge Convergence Service Platform using Image Recognition in Broadcasting Contents). It was also conducted using 2015 research grant from Kwangwoon University.

http://dx.doi.org/10.3837/tiis.2015.11.019 **ISSN : 1976-7277**

1. Introduction

High speed broadband networks and improvements in display technology have enabled multimedia streaming to become the most popular application in consumer electronics. Video traffic dominates Internet traffic on both fixed and mobile access throughout the world. Video streaming services, such as Netflix and YouTube account for the majority of real-time entertainment traffic [1].

Traditional streaming technologies completely download the video encoded at a particular video rate before playback can begin. Recently, streaming technologies are based on hypertext transport protocol (HTTP) due to several benefits that it offers. The benefits include, (1) the Internet infrastructure support, (2) firewall friendliness, and (3) the client does not have to maintain a session state on the server. Dynamic adaptive streaming over HTTP (DASH) [2] is an HTTP-based technique in which each video is divided into segments encoded at multiple rates. The video client requests a segment encoded at a particular video rate with an HTTP GET message, depending on the network conditions.

Adaptive bitrate (ABR) algorithms are responsible for the selection of the video bitrates of the segments. ABR algorithms try to maximize the quality of the video to improve the user's viewing experience. In this respect, algorithms strive to maximize the user experience by meeting conflicting video quality objectives in different environments. 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 [3][4]. Maximizing the video rate would risk rebuffering, whereas minimizing rebuffering would lead to low video quality. In an environment with variable throughput, selecting a higher feasible video rate may lead to high frequency of video rate switches. Note that the terms "buffer level" and "buffer size" are measured in seconds.

One way to pick video bitrates is to make an estimate of the future throughput from past observations [5-7]. Accurate estimation of the dynamics of throughput becomes an important challenge for the client. An inaccurate estimation may lead to selecting video bitrate that result in extensive rebuffering. To avoid interrupting playback, ABR algorithms add playback buffer occupancy as an adjustment parameter on top of throughput estimation to select video bitrates [8-10]. In [11], the authors proposed the video rate adaption algorithm that only selects the video rate by observing the client's playback buffer.

In this paper, we designed an ABR algorithm that selects the video rates based on both estimated throughput and buffer occupancy. The algorithm divides the playback buffer into set of buffer thresholds to select respective video rates. As the throughput increases, the algorithm increases the video rate when the buffer level increases above the threshold. As the throughput decreases, the algorithm gracefully decreases the video rate when the playback buffer drops below the threshold. When the throughput periodically fluctuates, the algorithm shows a stable response by absorbing video rate switching. We first analyze various throughput estimation methods and show that the method we have used in this

paper offers stable response to throughput fluctuations while maintaining a stable playback buffer. The goal of the proposed ABR algorithm is to maximize the performance of all three metrics (1) video rate (2) rebuffering rate and (3) switching rate to provide the users with a positive viewing experience. We show that our method offers a stable response to short term fluctuations and swiftly responds to large fluctuations in the throughput. We show that the algorithm provides viewers with a better quality of experience (QoE) for both constant bitrate (CBR) and variable bitrate (VBR) encoded video content.

2. Overview of Adaptive HTTP Streaming

2.1 HTTP Adaptive Streaming

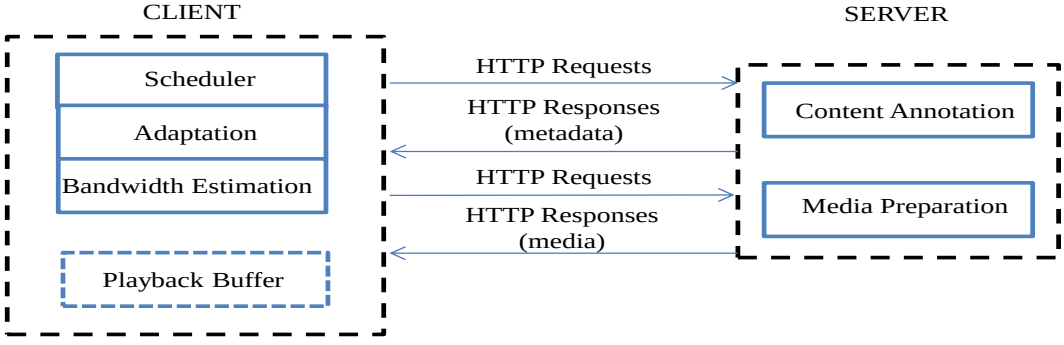

Fig. 1. HTTP Streaming Architecture

HTTP adaptive streaming works by monitoring network conditions in real time and adjusts the quality of the video stream accordingly without resetting the TCP connection. The basic model of adaptive HTTP streaming is shown in **Fig. 1**. Adaptive streaming requires multiple versions of the multimedia content to be stored at the server. At the server side, the *content annotation module* provides information about the characteristics of the stored multimedia content. The client initiates the request for the information about the stored content, which is known as metadata. In response to the request from the client, the server sends 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 *scheduler module* is responsible for scheduling the download of upcoming segments. The *bandwidth estimation module* estimates the throughput during the download of the segments. The *adaptation module* selects a suitable bitrate depending on the received metadata and system conditions such as throughput and the 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

Adaptation methods mainly select video rates based on the throughput and playback buffer.

Segment throughput is calculated as the ratio of segment size divided by the time it takes to download the segment. The selection of the video rate of the next segment is based on the instant throughput $T(i)$ of the last downloaded segment i. This selection keeps the buffer stable but results in a fluctuating video rate curve. A smoothed throughput measure is used to estimate the throughput $T^E(i+1)$ of the next segment to overcome the fluctuating video rate curve given by [5]:

$$T^E(i+1) = (1-\delta) \times T^E(i) + \delta \times T(i) \tag{1}$$

where δ is a weighing factor. This results in a late reaction to the change in throughput which can deplete the playback buffer. Further throughput estimates are computed from round trip time (RTT) [12]. Once the throughput has been estimated, clients pick the video rate $R(i+1)$ of the next segment $i+1$ [6].

$$R(i+1) = (1-\mu) \times T^E(i+1) \tag{2}$$

where μ is a value within the range [0, 0.5]. Thus, selecting the video bitrate depends on estimating the throughput.

In this paper, we compare our scheme with the algorithms proposed in [8] and [9]. The algorithm in [8] divides the streaming session into fast start and steady phase. In the fast start phase, it starts with downloading the lowest available video rate. During both phases, the buffer is divided into multiple ranges (0, B_{min}, B_{low}, B_{high}, B_{max}) where ($0<B_{min}< B_{low}< B_{high}< B_{max}$). Different decisions are taken to select the video rates when the buffer level stays in different ranges. In the steady state, the algorithm waits for the buffer level to reach predefined upper and lower buffer thresholds [B_{low}, B_{high}] before it decides to increase or decrease the video rate. It stays at the current video rate if the buffer level stays between the thresholds, to reduce the frequency of video rate switches. The algorithm tries to keep the buffer level close to the target interval B_{opt} where $B_{opt}=0.5(B_{low}+B_{high})$ to efficiently avoid playback interruptions. In [9], the algorithm selects a video rate less than the estimated throughput. In case of the increase in the throughput, the algorithm increases the video rate once the buffer level is larger than a pre-defined threshold. The algorithm increases the video rate by only one level at a time to mitigate the effects of throughput fluctuations and buffer drainage. It estimates the elapsed time to deplete the segments in the playback buffer and the expected number of segments that can be downloaded during the elapsed time. Based on the elapsed time and the expected downloadable segments, it tries to preserve a predefined minimum buffer length. In case of a drop in the throughput, it tries to consume the buffer to keep the next video rate comparable to the previous video rate since a large drop in quality level impairs the QoE.

The primary goal of ABR algorithms is to make the best decisions to optimize the viewer's quality of experience. Several studies have been done to understand the factors that affect user engagement. It has been shown that QoE depends on the rebuffering events and video rate [3][13]. Both factors are contradictory because maximizing the video rate

would risk extensive rebuffering. Moreover, it has been shown that the frequency of video rate switches impairs user engagement [4]. A rebuffering event has been identified as one of the most important parameters. Our focus in this work is to optimize all three conflicting metrics to improve the viewing experience and user engagement in response to both short term and large fluctuations in the throughput.

3. Throughput Estimation

In HTTP streaming, media segments are delivered by a series of consecutive HTTP request-response messages. As discussed in Section 2, the throughput is estimated before delivering the next segment. If the throughput, $T(t)$, of the last segment is used to decide the video rate, $R(t)$, of the next segment, short term fluctuations may cause frequent fluctuations in the video rate. To minimize the frequency of fluctuations, a smoothed throughput measure is used to stabilize the estimation over time. Smoothed throughput causes the client to react late to large fluctuations. ABR algorithm may overestimate the capacity due to the late response of the smoothed throughput estimation and pick a higher video rate. The greater the video rate, the larger will be the average segment size of video stream. The segments are downloaded into the buffer at a rate $T(t)/R(t)$. The playback buffer drains at a unit rate. If $T(t)/R(t) < 1$, the buffer level will decrease.

In this paper, we use the McGinely dynamic indicator (MDI) for the throughput estimation measure [14] which is given by:

$$T^E(i+1) = T^E(i) + \frac{T(i) - T^E(i)}{N \times \left(\frac{T(i)}{T^E(i)}\right)^4} \tag{3}$$

In Equation (3), $T^E(i)$ is the estimated throughput for i^{th} segment and $T(i)$ is the observed throughput over segment i and N is the tracking factor. The estimated throughput of segment i is modified to come up with $T^E(i+1)$. The numerator of the second term gives a sign, up or down and the power of 4 gives the calculation an adjustment factor which increases more sharply as the difference between the observed throughput of i^{th} segment and estimated throughput of segment i increases. This makes the size of the adjustment change logarithmically. In this paper, we use the value of tracking factor N equal to 1.

Fig. 2 shows the comparison of MDI with the following throughput methods: (1) instant throughput (IT), (2) moving average of the throughput of last 10 segments (MA) which is implemented in many commercial clients [15], (3) smoothed throughput using Equation (1) with δ=0.8 (ST-1) and (4) balancing the RTT-based and segment-based estimation by selecting an estimation period which is larger than RTT and smaller than the duration of a segment. The bandwidth is sampled every θ seconds and computed as [9]:

$$T^{sample} = \frac{K}{\theta} \tag{4}$$

where K is the amount of data downloaded during θ seconds. To smooth out the fluctuations in the network throughput, the estimated throughput is smoothed using a weighted moving average scheme as:

$$T^{E} = w \times T^{E} + (1-w) \times T^{sample} \tag{5}$$

where T^{E} is the previously estimated throughput and w is the weight factor for T^{sample}. The value of θ and w is set as 0.3 seconds and 0.875 respectively. We call it the smoothed throughput method 2 (ST-2) in the results. We compared the performance of the throughput estimation methods using network simulator 3 (ns-3). The server offers six discrete bitrates from 400kbps to 1400kbps with a step size of 200kbps. The size of each segment is 4s.

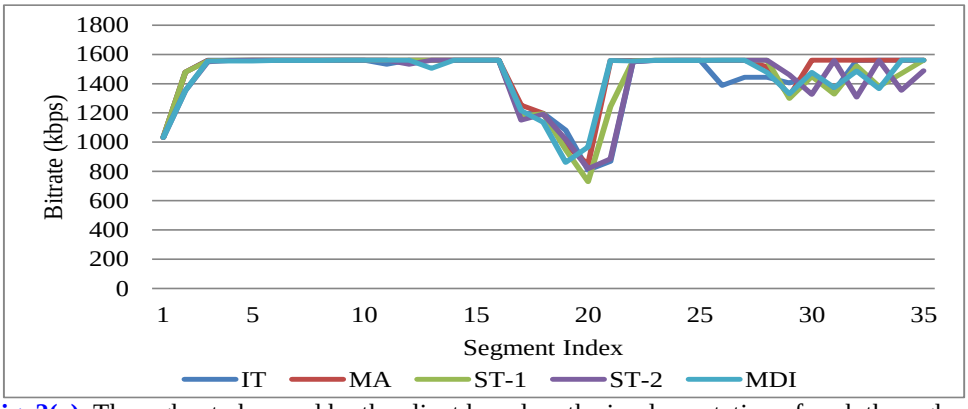

Fig. 2(a). Throughput observed by the client based on the implementation of each throughput estimation method

In **Fig. 2(a)**, the throughput curve represents $T(i)$. The video rate is picked by selecting the highest video rate less than $T(i)$. **Fig. 2(a)** shows the instant throughput observed by the HTTP client based on the implementation of each method. Initially the throughput rises above the highest available video rate. At the download of the 16th segment, there is a large drop in throughput followed by short-term fluctuations. **Fig. 2(b)** shows the bitrates achieved by the client during the implementation of various estimation methods which are decided by selecting the highest values less than the instant throughput. **Fig. 2(b)** shows that the throughput estimation using IT, ST-1 and ST-2 result in frequent video rate changes during both large scale and small scale fluctuations. The MDI scheme gradually drops the video rate from 1400kbps to 1000kbps during large drop in the throughput but shows a stable response during short-term fluctuations. The estimation using the MA method provides the most stable video rate response. **Fig. 2(c)** shows the resulting buffer level of each throughput estimation method. It shows that the selection of the video rates using IT and ST-1 result in a stable buffer whereas the selection of the video rates using ST-2 and MDI provide a gradual decrease of the buffer level. The MA reduces the buffer

level quickly because it tries to maintain a stable video bitrate and due to the late response to the throughput fluctuations.

Fig. 2 shows that the estimation method we use in this paper provides a balance between frequency of bitrate switches and stability of the buffer.

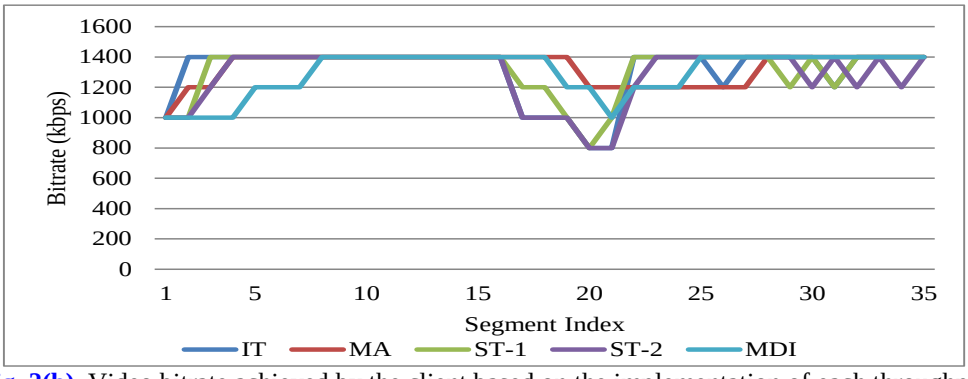

Fig. 2(b). Video bitrate achieved by the client based on the implementation of each throughput estimation method

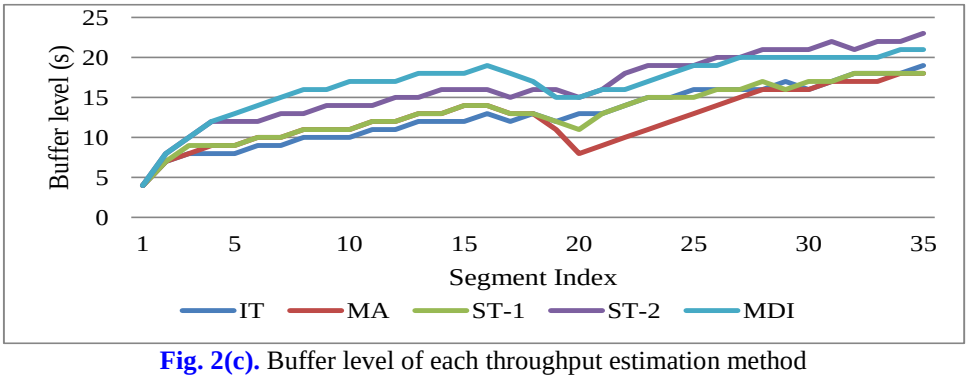

Fig. 2(c). Buffer level of each throughput estimation method

4. Proposed Scheme

In this section we present the proposed scheme that considers both the throughput and playback buffer to select video rates. In section 4-A, we discuss the design requirements; Section 4-B presents the system model; Section 4-C presents the adaptive bitrate algorithm.

4.1 Design Requirements

The main goal of the proposed scheme 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. Video rates and rebuffering events are important factors in improving the QoE [3][13]. In addition, the frequency of video rate switching has been found to annoy the viewer [4]. These factors

have been considered as important design goals by authors of multiple ABR algorithms [8-10][16]. We consider the following design goals for the proposed scheme:

1) Maximize the video rates
2) Minimize the rebuffering events
3) Minimize the frequency of video rate switches

In addition, we show that the proposed algorithm is stable in the face of short-term fluctuations and responds quickly to large fluctuations of throughput. We will evaluate the performance of the proposed algorithm for both CBR and VBR encoded video.

4.2 System Model

The HTTP client downloads a video stream that has been divided into multiple segments. All of the segments have an equal duration of τ sec. The video is stored in multiple discrete representations at the HTTP server. The set of representations available for the video stream is denoted by R. The client dynamically selects a video rate from the set R. R_{min} and R_{max} are the representations with the highest and lowest video rate in the set R. We denote video rates higher and lower than current video rate R by $R\uparrow$ and $R\downarrow$ respectively.

In this paper, segments are downloaded using a serial segment fetching method where the next segment is requested after the current segment has been completely downloaded. Data of τ seconds is added to the buffer once the segment is completely downloaded. The client starts playing the video after the first segment has been downloaded.

4.3 Adaptive Bitrate Algorithm

The algorithm's pseudo-code is provided in Algorithm 1. We consider that Algorithm 1 is invoked immediately after the download of segment i-1. The algorithm selects the representation for the download of the next segment i. The bitrate scheme in Algorithm 1 considers the buffer level and throughput together. We divide the streaming session into two phases of operation: the startup phase and the steady phase. The startup phase is when the buffer is building up from being empty, to be followed by the steady phase.

During the startup phase, as the buffer builds up from being empty, it carries little information 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. The algorithm selects the minimum available video rate R_{min} to download the first segment. The reason behind selecting the minimum video rate for the first segment is that the client does not initially have any information about the condition of the network. Plus, this approach reduces the delay after the client requests the video and before the client plays the video. For $B(i\text{-}1) < B_{LOW}$, the client switches to a higher video rate if $R\uparrow < \alpha_1 \times T(i\text{-}1)$. For $B(i\text{-}1) > B_{LOW}$, we increase to a higher video rate if $R\uparrow < \alpha_2 \times T(i\text{-}1)$ where α_1 and α_2 are the safety margins and $\alpha_1 < \alpha_2$. The proposed scheme continues to use the startup phase until any of the following conditions are violated: (i) $B(i\text{-}2) < B(i\text{-}1)$; or (ii), $R_{\text{startupphase}} > R_{\text{steadyphase}}$.

The motivation behind the startup phase is to quickly fill up the buffer without risking

underflow. Therefore, the first condition allows us to shift to the steady phase if there is a decrease in the buffer level. The second condition allows us to quit the startup phase if the steady phase suggests a higher video rate. Afterwards, we use steady phase to select the video rate of the next segment.

Algorithm 1: Adaptation Algorithm

if Startup phase conditions hold true
$\wedge\ B(i\text{-}2) < B(i\text{-}1)$
$\wedge\ R_{\text{startupphase}} > R_{\text{steadyphase}}$ ***then***
if $B(i\text{-}1) < B_{\text{LOW}}$ ***then***
if $R\uparrow < \alpha_1 \times T(i\text{-}1)$ ***then***
$R_k(i) = R\uparrow$
else
if $R\uparrow < \alpha_2 \times T(i\text{-}1)$ ***then***
$R_k(i) = R\uparrow$
else
if $B(i\text{-}1) < B_{\min}$ ***then***
$R_k(i) = R_{min}$
else if $R(i\text{-}1) \neq R_{\min}$ && $B(i\text{-}1) < B_k(i)$ && $R_k(i\text{-}1) > \alpha_3 \times T^E(i)$ ***then***
$R_k(i) = R\downarrow$
else if $R_k(i\text{-}1) \neq R_{\max}$ && $R\uparrow < \alpha_3 \times T^E(i)$ && $B(i\text{-}1) > B_k\uparrow$ && $T^E(i) > T^E(i\text{-}1)$ ***then***
$R_k(i) = R\uparrow$
else
$R_k(i) = R_k(i\text{-}1)$

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

$$B(i) = B(i-1) + \tau - [\tau \times \frac{R_k(i)}{T(i)}] \tag{6}$$

where $R_k(i)$ is the k^{th} video rate from the set R and $T(i)$ is the video throughput observed during the download of segment i. Equation (6) shows that if the video rate is greater than the throughput, the playback buffer gets drained. If each segment contains τ seconds of data then $C_k(\text{i})$, the size of the i^{th} segment is $\tau \times R_k$ bits. Given the throughput $T(i)$ and video rate $R_k(i)$, the change in buffer level during the download of i^{th} segment is equal to:

$$B^* = B(i) - B(i-1) = \frac{C_k(i)}{R_k(i)} - \frac{C_k(i)}{T(i)} \tag{7}$$

where the buffer fills with $C_k(i) / R_k(i)$ seconds of data and the buffer pushes $C_k(i) / T(i)$ seconds of data to the decoder. Equation (6) can now be written as:

$$B(i) = B(i-1) + B^{*} \tag{8}$$

If $R_k(i) > T(i)$, B^{*} becomes negative which means that the buffer is consumed at a faster rate than the rate at which the buffer fills, therefore, $B(i)$ will be less than $B(i\text{-}1)$. We assume that $T(i)$ cannot be less than $R_{min} = R_1$. We denote the change in the buffer level when $T(i) = R_{k\text{-}1}$ and the client overestimates the throughput and selects the next higher video rate R_k for the i^{th} segment as B_k^{*}. Assuming that, $T(i) = R_1$ and the client selects the next higher video rate R_2, then the change in buffer level is equal to:

$$B_2^{*} = \left| \frac{C_2(i)}{R_2} - \frac{C_2(i)}{T(i) = R_1} \right| \tag{9}$$

where $|x|$ returns the absolute value of x. There should be at least one segment (τ sec) available in the buffer for uninterrupted video playback. We denote $B_k(i)$ as the minimum buffer level occupancy for a client to select the k^{th} video rate for the i^{th} segment.

$$B_k(i) = \tau + \left| \sum_{m=2}^{k} B_m^{*} \right| = \tau + \left| \sum_{m=2}^{k} \frac{C_m(i)}{R_m} - \frac{C_m(i)}{R_{m-1}} \right| \tag{10}$$

Rearranging Equation (10)

$$B_k(i) = \tau + \left| \sum_{m=2}^{k} \frac{C_m(i) \times R_m - C_m(i) \times R_{m-1}}{R_m \times R_{m-1}} \right| \tag{11}$$

Equation (11) ensures that if the client selects the k^{th} video rate when at least $B_k(i)$ amount of buffer is available and the throughput drops to R_{min}, there will be τ seconds of buffer available at the end of the segment download.

Here, we describe the operation of steady phase. We consider following four cases:

1) The client selects $R_{\min}$ as the next video rate
2) The client selects the video rate of the next segment lower than the current segment
3) The client selects the video rate of the next segment higher than the current segment
4) The client selects the video rate of the next segment equal to the current segment

If the buffer level falls below $B_{min} = B_2(i)$, R_{min} is always selected. $B_2(i)$ is the minimum buffer occupancy to select the video rate $R_2(i)$. The reason is that it is of primary importance to avoid interruption of the playback. When the buffer level is below B_{min}, there is a high risk of buffer underflow in the presence of throughput fluctuations.

To select the k^{th} video rate, two conditions should be satisfied:

1) The buffer level should be higher than $B_k(i)$
2) $R_k(i) < \alpha_3 \times T^{E}(i)$

The client will select $R_k(i)$ only if the buffer level is greater than $B_k(i)$. This condition helps in avoiding buffer underflow in case the client overestimates the throughput or there is a sudden drop in the throughput. The condition of $R_k(i) < \alpha_3 \times T^E(i)$ uses a safety margin α_3 to compute the bitrate to avoid throughput overestimation.

Now, we consider the scenario when the throughput and the buffer level drops. We do not immediately react to this drop in the throughput; we stay at the current video rate. We stay at the current video rate until the buffer level drops below $B_k(i)$. This is because we can minimize the number of video rate switches, if we don't react to short-term fluctuations. Once the buffer level falls below $B_k(i)$, we continue to reduce the video rate until the condition $R_k(i) < \alpha_3 \times T^E(i)$ is satisfied.

Next, 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, three conditions should be satisfied:

1) $R\uparrow < \alpha_3 \times T^E(i)$
2) The buffer level should be greater than $B_k\uparrow$
3) $T^E(i) > T^E(i\text{-}1)$

The first condition makes sure that the higher selected video rate is below a certain percentage of the estimated throughput, to avoid the drainage of the buffer. $B_k\uparrow$ is the minimum buffer occupancy to select the video rate that satisfies the first condition. As the video rate cannot be adapted until the download of the next segment, in case of a sudden drop in throughput the second condition reduces the probability of a buffer underflow event. The third condition reduces the frequency of video rate switching 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.

When the conditions of switching up and switching down the video rate are not satisfied, we maintain the current video rate.

5. Performance Evaluation

To evaluate the performance of the proposed algorithm, we implemented it in ns-3. We compare the proposed method with the adaptive streaming schemes proposed in [8] and [9]. The topology implemented in this paper is shown in **Fig. 3**. The topology consists of an HTTP server, HTTP client and a pair of network elements. The link between the network elements is our bottleneck link. We vary the bottleneck link to analyze the performance of the algorithms under different network conditions. To achieve adaptive streaming, the HTTP server offers the client seven levels of representations to adapt the video rates. These video rates include 356, 500, 800, 1200, 1500, 2100 and 2400Kbit/s. The length of each segment and playback buffer is 4 and 60 sec, respectively. B_{LOW} is set to 30% of the buffer size. The safety margins are set to $(\alpha_1, \alpha_2, \alpha_3)=(0.5, 0.75, 0.9)$.

We implemented the adaptive algorithm for adaptive streaming proposed in [8] using the same settings as mentioned in that paper. We call the algorithm AAAS in our results. For the QoE-enhanced adaptive algorithm (QAAD) proposed in [9], the value of the

minimum buffer length is set to 5 sec because the segment size in our experiments is 4 sec. The rest of the settings are kept the same as mentioned in that paper.

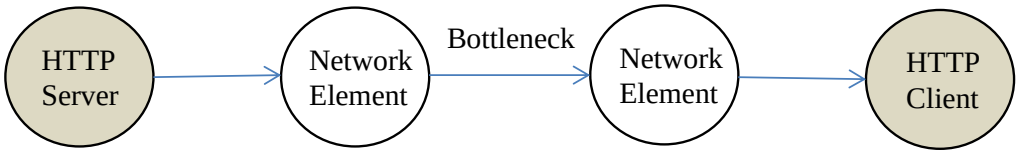

Fig. 3. Network Topology

5.1 Constant Bitrate Video

In this section we compare the performance of the proposed algorithm using a video encoded in constant bitrate (CBR). CBR encodes all the segments at an average video rate. As a result, all the segments sizes will be uniformly identical in a stream of a given rate. The value of $B_k(i)$ will stay constant througout the streaming session since the sizes of segments are identical in a stream of a given video rate.

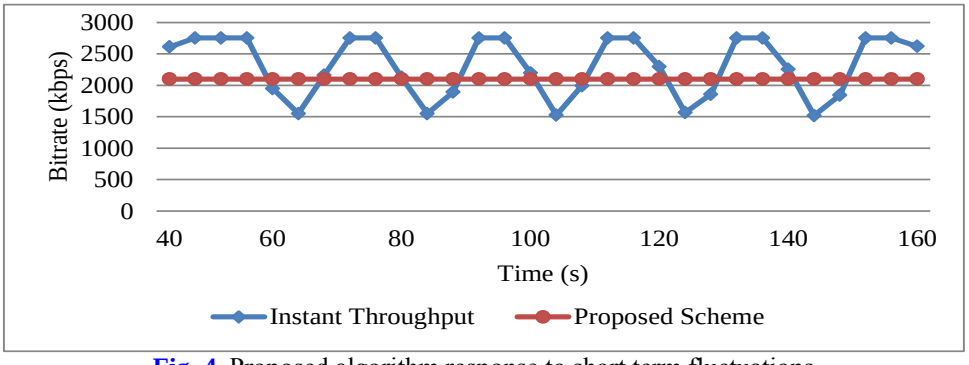

Fig. 4. Proposed algorithm response to short term fluctuations

Fig. 4 demonstrates the video rate selected by the proposed scheme under a short-term fluctuation scenario. The plotted graph shows the response of the proposed scheme while operating in the steady phase. It shows that the proposed scheme is stable in response to the short term fluctuations. The reason behind a stable response is that the buffer distance between $B_k(i)$ and $B_{k+1}(i)$ provides a cushion, and absorbs video rate switching.

In a bitrate adaptation, one of the most important cases is a sudden drop in throughput. **Fig. 5** shows that the throughput drops at 72 seconds. As the throughput drops, the proposed scheme switches down the video rate gradually. Like the proposed scheme, QAAD scheme drops the video rate gradually. The AAAS scheme is the most conservative of all the schemes. The reason behind this conservative approach is that it waits for the playback buffer to cross a given threshold before stepping up or stepping down the video rate. This results in stable behavior but at the cost of lower video quality.

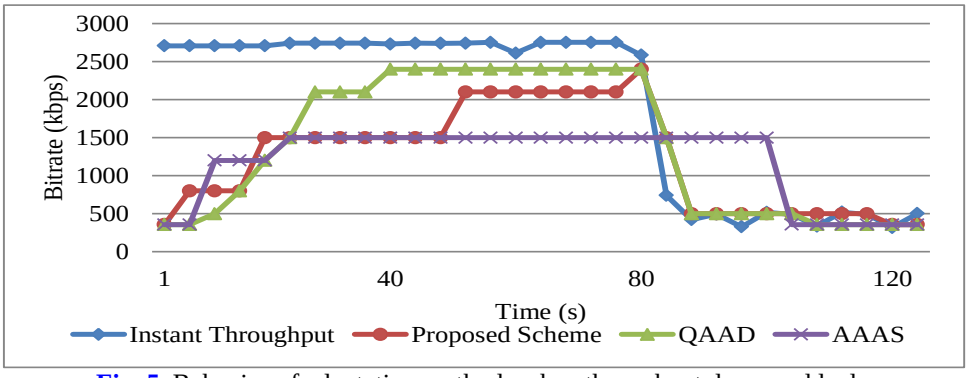

Fig. 5. Behavior of adaptation methods when throughput drops suddenly

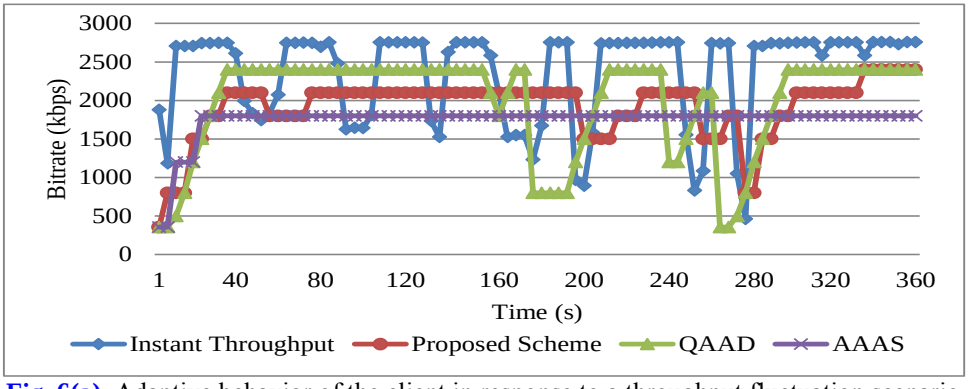

Fig. 6(a). Adaptive behavior of the client in response to a throughput fluctuation scenario

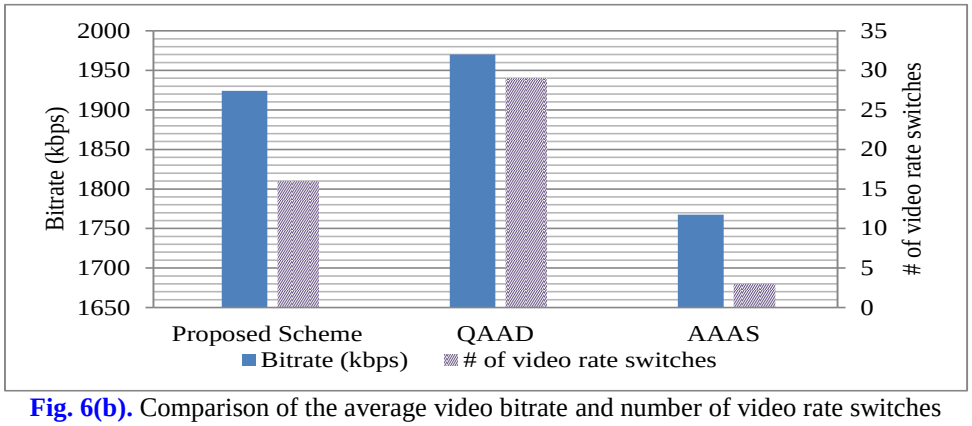

Fig. 6(b). Comparison of the average video bitrate and number of video rate switches

Fig. 6(a) shows the comparison of the schemes when throughput is fluctuating. It shows that the AAAS has a stable response to the spikes. The QAAD has a stable response from 40 to 160 sec but at the expense of consumption of the buffer. After 160 sec, as the buffer gets depleted, it switches video rate down. As the QAAD tries to stay at a higher video rate

at the expense of consuming buffer, there is a high risk of buffer underflow if the throughput drops suddenly. At 280th sec, the QAAD experiences buffer underflow since it does not secure the buffer. **Fig. 6(b)** shows that the proposed scheme achieves a higher video rate than the AAAS and less frequent bitrate changes compared to the QAAD. The QAAD scheme achieves a slightly higher video rate as compared to the proposed scheme but at the expense of buffer underflow events and a higher frequency of video rate switches. The AAAS is the most stable of all three schemes but achieves an average video rate of 170kbps less than the proposed scheme. In addition, the proposed scheme keeps the bitrate difference between any two consecutive segments small. This has less of a negative effect on the viewing experience during the rate switching event [17].

Fig. 7 shows the adaptive behavior of the client to the gradually decreasing throughput scenario. The **Fig. 7** shows that the QAAD maintains a high video rate from 40 to 220 sec but after the buffer is consumed it switches the video rate down as soon as the throughput decreases. The proposed scheme achieves a high average video rate without risking buffer underflow. Furthermore, as throughput drops gradually, the proposed scheme ensures that a small drop in throughput doesn't result in unnecessary stepping down of the video rate.

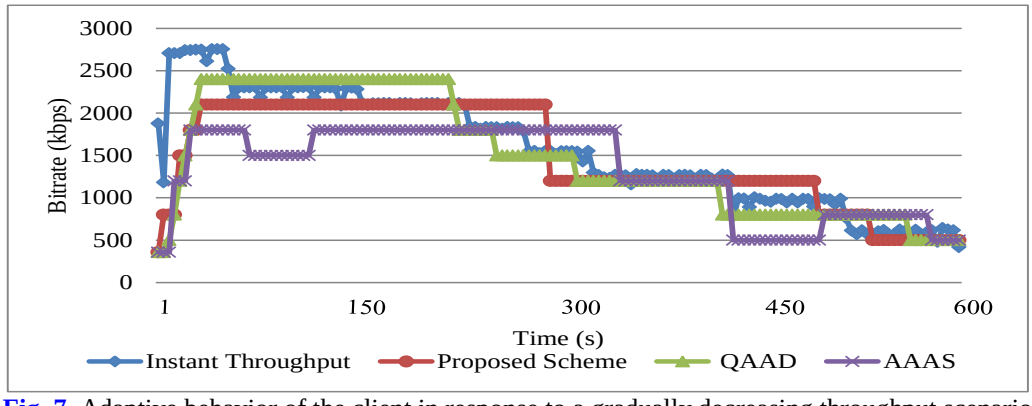

Fig. 7. Adaptive behavior of the client in response to a gradually decreasing throughput scenario

5.2 Variable Bitrate Video

Most of the streaming services encode videos in VBR where static scenes are encoded with fewer bits and active streams with more bits. This allows flexible and efficient use of bits. In VBR, video is encoded at an average video rate and the instantaneous video rate varies around the average rate. **Fig. 8** shows the segment sizes of video encoded at 450kbps with an average segment size of 1800kbits. Given the variation of segment sizes, the buffer dynamics discussed in Section 4 are reconsidered under VBR. As the size of each segment is different and $B_k(i)$ depends on segment size, the value of $B_k(i)$ will change every time a segment is downloaded. This makes the video rate change frequently which affects the QoE. Furthemore, for a given throughput a segment of a larger size will take more time to get downloaded, hence will consume more video in the buffer than a smaller segment.

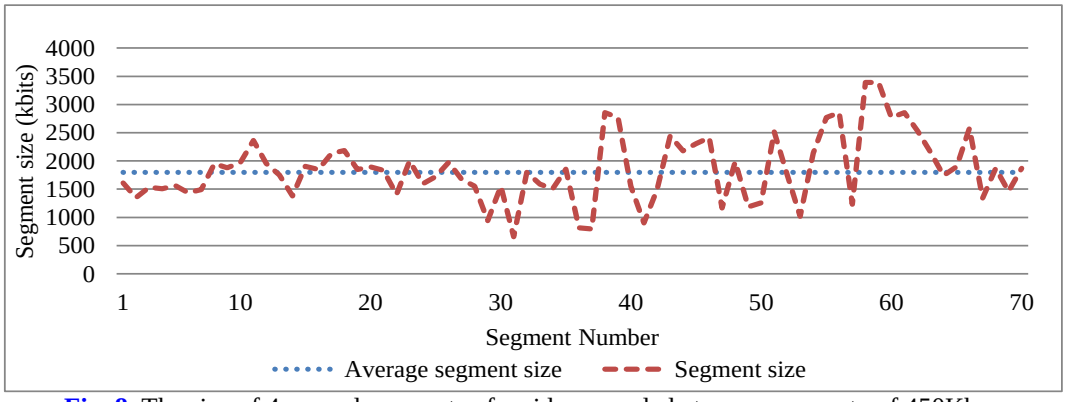

Fig. 8. The size of 4 second segments of a video encoded at an average rate of 450Kbps

To this end, we take the average of the next 10 segment sizes and calculate $B_k(i)$ after every 10 segments based on their average sizes. For VBR, we calculate $B_k(i)$ using:

$$B_k(i) = \tau + \sum_{m=2}^{k} \frac{\overline{C}_m \times R_m - \overline{C}_m \times R_{m-1}}{R_m \times R_{m-1}} \tag{12}$$

where $\overline{C}_m$ is the average of every 10 segment sizes. Equation (12) makes sure that $B_k(i)$ gets its value recalculated after every 10 segments. If the upcoming segments are larger, the buffer thresholds to select a given video rate will be greater than when upcoming segments are smaller. If we select the average segment size based on more than 10 upcoming segments, it might not correctly depict the segment size trend whereas calculating $B_k(i)$ based on fewer segments will result in a higher frequency of video rate switches. For VBR, the HTTP server offers 4 levels of representations for the client to adapt the video rates. These video rates include 450, 850, 1500, 2500Kbit/s.

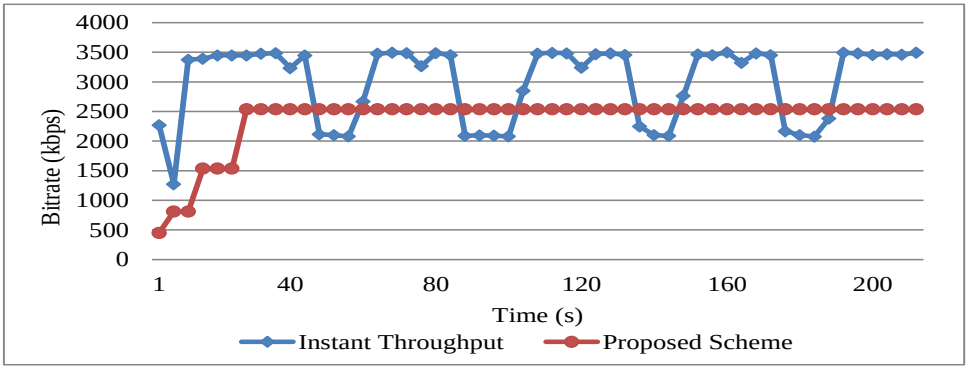

Fig. 9. Response of the VBR-enabled proposed scheme to small drop in throughput

Fig. 9 shows the response of the proposed VBR-enabled scheme to short term fluctuations. It shows that the proposed scheme is stable in the face of short term throughput fluctuations while it maintains high video rate.

Another important property of an adaptive scheme is that it should have a swift response to large fluctuations. **Fig. 10(a)** shows the adaptive bitrate behavior of the client to large fluctuations. Both the AAAS and QAAD algorithms experience a buffer underflow event while downloading the 58th segment. **Fig. 8** shows that the 58th segment is encoded with a higher bitrate and its segment size is twice the average segment size. Both AAAS and QAAD try to maintain a higher video rate at the expense of buffer consumption. The sudden drop in the throughput and the need to maintain a higher video rate to download a segment that was encoded at a larger size contributed to the buffer underflow. The proposed scheme on the other hand, reacts swiftly to large throughput fluctuations to avoid buffer underflow. Furthermore, the proposed VBR-enabled scheme varied $B_k(i)$ with segment size to provide a cushion to the buffer oscillations due to variable segment sizes. **Fig. 10(b)** shows that the proposed and QAAD scheme show similar performance with respect to the average video rate and number of video rate switches. AAAS scheme shows a stable response to the fluctuations in throughput but at the expense of video rate.

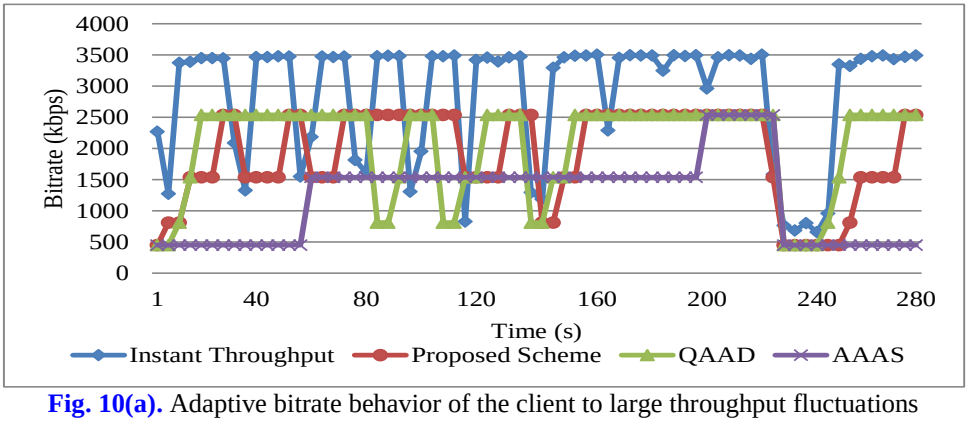

Fig. 10(a). Adaptive bitrate behavior of the client to large throughput fluctuations

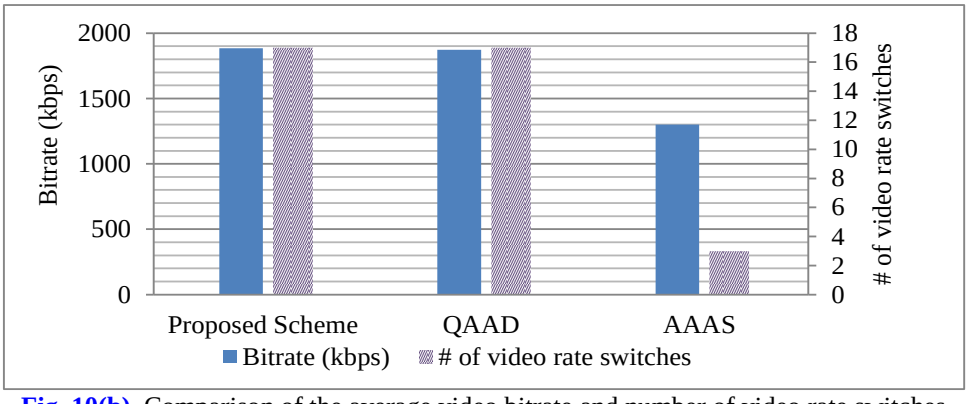

Fig. 10(b). Comparison of the average video bitrate and number of video rate switches

Fig. 11 shows the behavior of the client in response to a gradually decreasing throughput scenario. It shows that all schemes gradually drop the video rate. All schemes ensure that a small drop in throughput doesn't result in an unnecessary stepping down of the video rate.

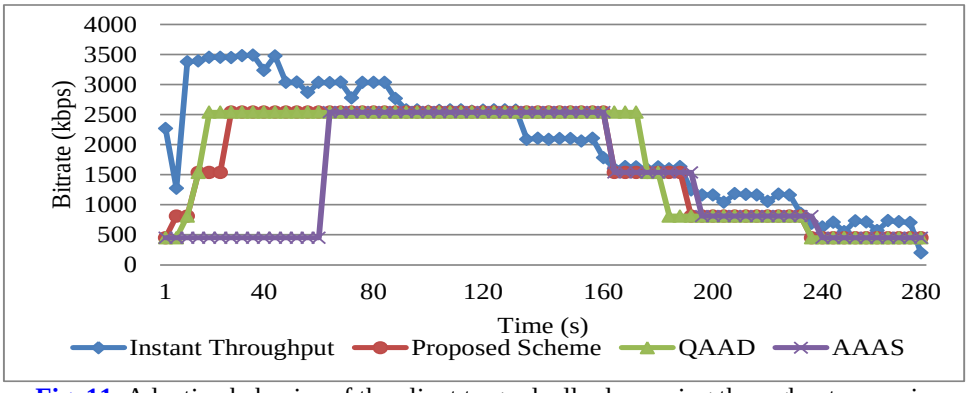

Fig. 11. Adaptive behavior of the client to gradually decreasing throughput scenario

Fig. 12 shows the plot of the sudden drop in the available throughput. There is a drop in the throughput around the 70th second. The plot shows that the client drops the video rates gradually while implementing the proposed and QAAD schemes. With the AAAS scheme, there is a sudden drop in video rate from 2500 to 450 kbps. In the implementation of the AAAS scheme, the client experiences a playback interruption. The reason behind it is that the scheme does not change the video rate until the buffer level crosses the lower threshold. In case of a large drop in throughput, the client stays at the current video rate at the expense of the buffer consumption which results in the playback interruption.

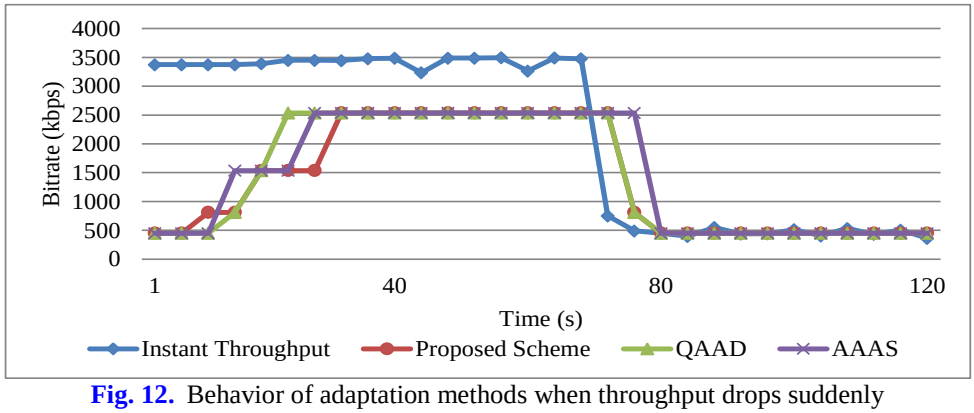

Fig. 12. Behavior of adaptation methods when throughput drops suddenly

The results show that the proposed VBR-enabled scheme achieves high a video rate without taking unnecessary risks. The scheme exhibits a stable response to short term fluctuations and reacts swiftly to large throughput fluctuations. In addition, the selection of

buffer thresholds based on segment sizes helps in absorbing the buffer oscillations due to variable segment sizes.

6. Conclusion

HTTP has been widely used for video streaming applications over the Internet. Video rate adaptation techniques are used to adapt to the varying network resources of the Internet. First we evaluated different throughput estimation methods and showed that the method we used in this paper is stable in the face of short term throughput fluctuations and maintained a stable buffer under throughput fluctuations. In this paper, we proposed an adaptive bitrate streaming algorithm to improve the viewing experience of the multimedia streaming applications. The proposed algorithm guarantees QoE by achieving a high video rate and minimizing the frequency of changes in video quality while preventing interruption in playback. We then showed that the proposed algorithm improves the viewing experience for both CBR and VBR video content by taking segment size into consideration.

References

[1] Sandvine, “Global Internet Phenomena Report 1H,” 2014. https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/1h-2014-global-internet-phenomena-report.pdf

[2] ISO/IEC 23009-1:2012, Information technology -Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats. Article (CrossRef Link)

[3] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang, "Understanding the impact of video quality on user engagement," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 362-373, August, 2011. Article (CrossRef Link)

[4] P. Ni, R. Eg, A. Eichhorn, C. Griwodz, and P. Halvorsen, "Flicker effects in adaptive video streaming to handheld devices," in *Proc. of ACM International Conference on Multimedia*, pp. 463-472, November, 2011. Article (CrossRef Link)

[5] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," in *Proc. of ACM Conference on Multimedia Systems*, pp. 157-168, February, 2011. Article (CrossRef Link)

[6] T. C. Thang, Q. D. Ho, J. W. Kang, and A. T. Pham, “Adaptive streaming of audiovisual content using MPEG DASH,” *IEEE Transactions on Consumer Electronics*, vol. 58, no. 1, pp. 78-85, February, 2012. Article (CrossRef Link)

[7] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate adaptation for adaptive HTTP streaming," in *Proc. of the ACM conference on Multimedia systems*, pp. 169-174, February, 2011. Article (CrossRef Link)

[8] K. Miller, E. Quacchio, G. Gennari, and A. Wolisz, "Adaptation algorithm for adaptive streaming over HTTP," in *Proc. of the IEEE Packet Video Workshop*, pp. 173-178, May, 2012. Article (CrossRef Link)

[9] D. Suh, I. Jang, and S. Pack, “QoE-enhanced adaptation algorithm over DASH for multimedia streaming,” in *Proc. of IEEE Conference on Information Networking*, pp. 497-501, February, 2014. Article (CrossRef Link)

[10] H. T. Le, D. V. Nguyen, N. P. Ngoc, A. T. Pham, and T. C. Thang, "Buffer-based bitrate adaptation for adaptive HTTP streaming," in *Proc. of IEEE Conference on Advanced Technologies for Communications*, pp. 33-38, October, 2013. Article (CrossRef Link)
[11] T. Y. Huang, R. Johari, and N. McKeown, "Downtown abbey without the hiccups: Buffer-based rate adaptation for http video streaming," in *Proc. of the ACM SIGCOMM Workshop on Future Human-Centric Multimedia Networking*, pp. 9-14, August, 2013. Article (CrossRef Link)
[12] T. C. Thang, T. L. Hung, H. X. Nguyen, A. T. Pham, J. W. Kang, and Y. M. Ro, "Adaptive video streaming over HTTP with dynamic resource estimation," *Journal of Communications and Networks*, vol. 15, no. 6, pp. 635-644, 2013. Article (CrossRef Link)
[13] Y. Liu, S. Dey, D. Gillies, F. Ulupinar, and M. Luby, "User Experience Modeling for DASH Video," in *Proc. of the IEEE Packet Video Workshop*, pp. 1-8, December, 2013. Article (CrossRef Link)
[14] John R. McGinley, "McGinley Dynamics," *Market Technicians Association Journal*, issue 48, pp. 15-18, 1997. [Online]. Available: https://www.mta.org/eweb/dynamicpage.aspx?webcode=journal-technical-analysis-1997-summer
[15] T. Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, timid, and unstable: picking a video streaming rate is hard," in *Proc. of ACM conference on Internet Measurement Conference*, pp. 225-238, November, 2012. Article (CrossRef Link)
[16] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo, "Elastic: a client-side controller for dynamic adaptive streaming over http (dash)," in *Proc. of the IEEE Packet Video Workshop*, pp. 1-8, December, 2013. Article (CrossRef Link)
[17] T. C. Thang, H. T. Le, A. T. Pham, and Y. M. Ro, "An evaluation of bitrate adaptation methods for HTTP live streaming," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 4, pp. 693-705, 2014. Article (CrossRef Link)

Waqas ur Rahman received the B.S. degree in Electrical Engineering from COMSATS Institute of Information Technology, Islamabad, Pakistan and M.S. degree in Computer Engineering from University of Engineering and Technology, Taxila, Pakistan in 2009 and 2012 respectively. Currently, he is pursuing Ph.D. degree in Department of Electronics and Communications Engineering at Kwangwoon university under the guidance of Prof. Kwangsue Chung. His research interests include QoS, QoE support, and rate control for video communications over wired/wireless networks.

Kwangsue Chung received the B.S. degree from Hanyang University, Seoul, Korea, M.S. degree from KAIST (Korea Advanced Institute of Science and Technology), Seoul, Korea, Ph.D. degree from University of Florida, Gainesville, Florida, USA, all from Electrical Engineering Department. Before joining the Kwangwoon University in 1993, he spent 10 years with ETRI (Electronics And Telecommunications Research Institute) as a research staff. He was also an adjunct professor of KAIST from 1991 to 1992 and a visiting scholar at the University of California, Irvine from 2003 to 2004. His research interests include communication protocols and networks, QoS mechanism, and video streaming. Dr. Chung is a senior member of IEEE.