TELKOMNIKA, Vol.14, No.3, September 2016, pp. 904~915
ISSN: 1693-6930, accredited A by DIKTI, Decree No: 58/DIKTI/Kep/2013
DOI: 10.12928/TELKOMNIKA.v14i3.3517

Towards Smooth and High-Quality Bitrate Adaptation for HTTP Adaptive Streaming

Lihong Geng[1], Liang Pan[2], Yiqiang Sheng[3], Zhichuan Guo*[4]
[1,2,3,4]National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China
[1]University of Chinese Academy of Sciences, Beijing 100190, China
*Corresponding author, e-mail: guozc@dsp.ac.cn

Abstract

Although HTTP adaptive streaming has been well documented for the cost-effective delivery of video streaming, it is still a great challenge to play back video smoothly with high quality under the fluctuating network conditions. In this paper, we proposed a novel bitrate adaptation algorithm for HTTP adaptive streaming. Our algorithm employed two approaches for throughput estimation and bitrate selection, which was evaluated on our testbed (a fully functional HTTP Live Streaming system) over a network, emulated using DummyNet. First, the throughput estimation method, based on the prediction of the difference between the estimated and instantaneous throughputs, was observed to respond smoothly to short-term fluctuations and rapidly to large fluctuations. Second, the bitrate selection algorithm, based on piecewise functions to define the variation range of the current bitrate, was found to result in smoother changes in quality with a higher average quality. The results of our experiments demonstrated the prospects of our bitrate adaptation algorithm for HTTP adaptive streaming.

Keywords*: throughput estimation, bitrate selection, HLS*

1. Introduction

With the diffusion of new video-enabled devices and faster Internet connections, video traffic has come to dominate Internet traffic. Several video streaming protocols have been proposed for the delivery of video content. Traditional streaming protocols, such as real time streaming protocol (RTSP), control the video transmission rate directly; however, such protocols are difficult to deploy because a specialized streaming server is required [1]. By contrast, hypertext transfer protocol (HTTP [2]) adaptive streaming has become a cost-effective and popular option because it reuses the existing Internet infrastructure, provides network address translation (NAT) friendliness and is allowed by most firewalls [3]. Currently, one of the most prominent approaches is HTTP Live Streaming (HLS), as proposed by Apple Inc.

In general, an HTTP adaptive streaming server knows little information about the client, it is the client's responsibility to make decisions regarding the selection of appropriate alternatives to maintain a good quality of experience (QoE [4]). To provide the highest possible video quality, an adaptation algorithm must appropriately estimate the available throughput. Overestimation of the throughput may lead to buffer freezes, whereas underestimation of the throughput may lead to buffer overflow. Furthermore, to provide users with a smoother video quality, a good bitrate selection algorithm is desirable. If the bitrate is selected based only on the estimated throughput, then abrupt changes in quality will occur when the available throughput decreases or increases dramatically. How to fulfill both requirements discussed above is a key research problem of HTTP adaptive streaming [5]. Researchers [6-12] have reported various algorithms for throughput estimation and bitrate selection.

Regarding throughput estimation, in [6], the authors used the running average of the measured throughput to estimate the throughput. However, this method exhibited a slow response to large changes in throughput and thus had a tendency to suffer buffer underflow. In [7], the degree of fluctuation of the throughput difference was used to dynamically control the weighting coefficient of the method presented in [8] for throughput estimation. Although this method showed superiority in detecting large changes in throughput, it failed to smoothly estimate the throughput in the case of short-term fluctuations and thus incurred redundant

Received February 26, 2016; Revised June 8, 2016; Accepted June 29, 2016

fluctuations in bitrate. In [9], an exponential function was adopted to dynamically control the weighting coefficient of the method presented in [6]. This method offered smooth estimation in the case of short-term fluctuations at a relatively high throughput. However, at low throughput, the estimated throughput of this method was sensitive to such fluctuations.

Regarding bitrate selection, in [10], the authors achieved the smallest changes in video quality and the fewest interruptions by preserving the minimum buffer length. However, when the mismatch between the available throughput and the video bitrate was significant, this algorithm appeared to suffer from abrupt changes in bitrate. Additionally, it required too much time to adjust the quality in accordance with the available throughput. A quick boot algorithm was proposed to solve this type of problem in [11]. Furthermore, the authors used a fixed-interval buffer model to keep the bitrate unchanged whenever the buffer size was within a preset interval. Although this method reduced the number of changes in bitrate, it required a large buffer to cope with long-term variations in the available throughput. In [12], the authors only used the buffer to effectively reduce the rebuffer rate, suggesting that the current buffer level was sufficient for an adaptation algorithm in the steady state. In addition to the current buffer, we emphasize the importance of the variation range of the current bitrate in an adaptation algorithm.

In this paper, we present a novel bitrate adaptation algorithm for HTTP adaptive streaming. Our bitrate adaptation algorithm includes a throughput estimation method and a bitrate selection algorithm. The contributions of the paper are three-fold. First, with the intent of reacting smoothly to short-term fluctuations and quickly to large fluctuations, a novel throughput estimation method is proposed based on predicting the difference between the estimated and instantaneous throughputs. Second, to provide users with a high and relatively smooth video quality, we propose an innovative bitrate selection algorithm based on piecewise functions to define the variation range of the current bitrate. Finally, to verify the performance of our bitrate adaptation algorithm, we report the implementation of a fully functional HLS system.

The paper is organized as follows. In Section 2, we first provide an overview of HLS. Section 3 describes our bitrate adaptation algorithm. In Section 4, the experiments are discussed in detail. Finally, conclusions and future work are addressed in Section 5.

2. Overview of HLS

HLS is a media streaming protocol based on HTTP implemented by Apple. It is widely used by video streaming providers for its easy deployment, dynamical adaptability and strong penetrability [13]. Conceptually, A HLS system has three parts: a server, a distribution system, and a client. The server is responsible for encoding the input streams as MPEG-4 (AAC audio and H.264 video), encapsulating them in MPEG-2 TS format, and preparing the encapsulated media for distribution. The distribution system is a standard web server which is responsible for accepting and responding client's requests. The server and distribution system are integrated as HLS server in this paper. The client is responsible for choosing the appropriate media to request, downloading them, and then reassembling them to present.

Adaptation is an important part of HLS [14]. On the HLS server side, the original video content is encoded into multiple alternatives (versions) at different bitrates. Then, each alternative is further partitioned into a series of small segments (chunks) with the same duration. Simultaneously, the characteristics of each alternative such as bitrate, coding, and resolution are recorded in the manifest file with the, m3u8 extension. All the media segments and manifest files are stored in the distribution system. On the HLS client side, according to the status of the terminal/network, the most appropriate alternative is downloaded through the sending of consecutive HTTP requests. Therefore, the client eventually gets the whole video consisting of the segments at different bitrates.

3. The Proposed Bitrate Adaptation Algorithm

In this section, our bitrate adaptation algorithm, which consists of throughput estimation and bitrate selection, is described in detail. As stated above, multiple alternatives of a segmented sequence at different bitrates are stored in the server. Each segment contains τ seconds of playback. The set of bitrates for m different video qualities is denoted by

$R=\{R_1,R_2,\cdots\cdots R_m\}$, where R_i is the ith bitrate in R. We assume that $R_i<R_j$ if $i<j$. The client downloads the segments in chronological order, and all downloads are non-preemptive, i.e., the download of segment i cannot begin until segment $i-1$ has been completely downloaded. After each fragment is downloaded, our throughput estimation method is invoked to estimate the available throughput. Then, the proposed bitrate adaptation algorithm will run the bitrate selection algorithm to select the most appropriate bitrate for the next segment. The details of the algorithm are described below.

3.1. Throughput Estimation Method

Throughput estimation is one of the most crucial concerns in adaptive streaming [11]. Usually, the available throughput is calculated by dividing the data size of a segment by its delivery duration, as denoted by:

$$T_i=\frac{S_i}{t_i} \tag{1}$$

Where S_i and t_i are the size and download time, respectively, of the ith segment and T_i is the available throughput for the ith segment. Below, we call T_i the instantaneous throughput. The simplest method to estimate the available throughput is merely to use the instantaneous throughput. This method yields a stable buffer level; however, the video quality fluctuates. A smoothing strategy was adopted in [15] to solve the problem of fluctuations. However, this method reacts slowly to large fluctuations, which may result in playback freezes. In [9], a dynamic weighting coefficient was applied in the smoothing method to solve this problem. Although this method possesses the advantages of both methods discussed above, it still yields fluctuating estimates when the available throughput is low.

The goal of our method is to achieve a throughput estimation that 1) is stable in the case of short-term fluctuations and 2) reacts quickly to large fluctuations. The underlying principle of our method is as follows:

$$T_{i+1}^e=T_i+D_i^e \tag{2}$$

Where T_{i+1}^e is the estimated throughput for the $(i+1)$th segment and D_i^e is the predicted difference value. A precise prediction of D_i^e yields a precise T_{i+1}^e. D_i^e is defined by (3) and (4) in our method:

$$D_i=T_i^e-T_i \tag{3}$$

$$D_i^e=D_i*\delta \tag{4}$$

Where D_i reflects the degree of fluctuation of the throughput and δ is a correction term that takes values on the interval $(0,1)$. For short-term fluctuations, D_i is small and δ should be closer to 1 to bring T_{i+1}^e closer to T_i^e, resulting in a smoother estimation. For large fluctuations, D_i is large and a δ closer to 0 is required to bring T_{i+1}^e closer to T_i to allow a more rapid response to large fluctuations. In summary, the value of the parameter δ is strongly related to D_i in our analysis. The control functions relating δ and D_i are designed as shown in (5) and (6):

$$\rho=\frac{|D_i|}{T^{nor}} \tag{5}$$

$$\delta=\frac{1}{1+Ne^{M(\rho-\rho_0)}} \tag{6}$$

Where ρ is the normalization value of D_i and T^{nor} is a normalization factor; M and ρ_0 are the steepness and midpoint of the control function (6), respectively; N is a function of the buffer occupancy. In our method, we assume that the fluctuations are short-term when D_i is less than 10% of T^{nor} (i.e., $\rho < 0.1$) and the fluctuations are large when D_i is larger than 20% of T^{nor} (i.e., $\rho > 0.2$). M and ρ_0 need to be set properly to make sure that a ρ smaller than 0.1 will yield a δ closer to 1, in other words, our method gets a smooth estimation for short-term fluctuations; by contrast, a ρ larger than 0.2 will obtain a δ closer to 0, suggesting that our method quickly responds to large fluctuations. Furthermore, in case of buffer draining-up, N will dominate the speed of control in (6) when the buffer level is lower than the minimum threshold. It is defined as expressed in (7) and (8):

$$\mu = \frac{B_{cur} - B_{min}}{B_{max} - B_{min}} \tag{7}$$

$$N = \begin{cases} \frac{1}{\mu} & \mu > 0 \\ 100 & \mu \le 0 \end{cases} \tag{8}$$

Where B_{cur} is the current buffer level which is measured in seconds, B_{min} and B_{max} are the minimum and maximum buffer thresholds, respectively, and μ is the buffer occupancy.

3.2. Bitrate Selection Algorithm

In this section, given the estimated throughput as described above, we propose an innovative bitrate selection algorithm that considers the variation range of the current bitrate to achieve a high and relatively smooth video quality. Besides, buffer underflow and overflow are considered in our algorithm.

To achieve our goal, we studied the Just Noticeable Difference (JND). The JND [16] is defined to describe the smallest perceptual difference/change between two stimuli (e.g., two versions of a video) that could be detected by human perception; this work concluded that if the video versions were equally spaced by 3 JND units, a separation that yielded no obvious difference in practice, the typical number of versions was 4 to 7. On the basis of this conclusion, the bitrates of the Football and Soccer sequences were spaced by 1.5 JND units in [17], resulting in bitrates of 3000, 1495, 1038, 773, 640, 550, 427, 322, 260, and 222 Kbps; this list of bitrates, which belonged to fast-motion group, should thus also be safe for slow-motion group. Based on these conclusions, two piecewise functions are designed to determine the safe variation range of the current bitrate. Equations (9) and (10) present these functions.

$$R_{chg\_up} = \begin{cases} 100 & R_{cur} < R_{LowTh} \\ 200 & R_{LowTh} \le R_{cur} < R_{MidTh} \\ 400 & R_{MidTh} \le R_{cur} < R_{HigTh} \\ 1400 & R_{cur} \ge R_{HigTh} \end{cases} \tag{9}$$

$$R_{chg\_down} = \begin{cases} 100 & R_{cur} \le R_{LowTh} \\ min(R_{cur} - R_{LowTh}, 200) & R_{LowTh} < R_{cur} \le R_{MidTh} \\ max(R_{cur} - R_{MidTh}, 200) & R_{MidTh} < R_{cur} < R_{HigTh} \\ max(R_{cur} - R_{HigTh}, 400) & R_{cur} \ge R_{HigTh} \end{cases} \tag{10}$$

Where R_{cur} is the current bitrate; R_{LowTh}, R_{MidTh} and R_{HigTh} are three bitrate thresholds that satisfy $R_{LowTh} < R_{MidTh} < R_{HigTh}$; the functions min and max return the minimum and maximum value, respectively, between their two inputs; and the return values, R_{chg_up} and R_{chg_down}, define

the safe variation range of the current bitrate. If the available throughput is increasing, equation (9) is used to determine the variation range, i.e., R_{chg_up}. By contrast, if the available throughput is decreasing, equation (10) is used to determine R_{chg_down}. We convert $R_{chg\_up} / R_{chg\_down}$ into the corresponding bitrate index with equation (11):

$$Index_{chg} = f(R_{chg\_up} / R_{chg\_down}, R_{cur}) \quad (11)$$

Where f is a conversion function and $Index_{chg}$ is the safe variation range of the bitrate index corresponding to $R_{chg\_up} / R_{chg\_down}$.

Based on the safe variation range of the current bitrate as determined above, the procedures performed in our algorithm are described in algorithm 1. Here, $RIndex_{best}$ is initialized as the bitrate index with the highest possible value that is lower than the current estimated throughput; in other words, it satisfies:

$$R_{RIndex_{best}} = max\{R_k \mid R_k < T^e, R_k \in R\} \quad (12)$$

$RIndex_{last}$ is the bitrate index of the last segment. $DifIndex$ is the absolute value of D_{ori} which is equal to the difference between $RIndex_{best}$ and $RIndex_{last}$. B_{min}, B_{mid} and B_{max} are buffer thresholds, measured in seconds, which satisfy $B_{min} < B_{mid} < B_{max}$.

The input arguments of our algorithm include the instantaneous throughput (T), the estimated throughput (T^e), the current buffer level (B_{cur}) and the bitrate index of the last segment ($RIndex_{last}$). The output argument is the bitrate index for the next segment ($RIndex_{next}$).

When $D_{ori} \geq 0$, an equal or higher bitrate can be requested for the next segment. The current buffer level is considered in this case. If $B_{cur} \leq B_{mid}$, it is not wise to increase the bitrate immediately because the buffer level is still insufficient. Thus, we leave the bitrate unchanged ($RIndex_{next} = RIndex_{last}$). If $B_{cur} > B_{mid}$, then it is safe to increase the bitrate. As described above, it is necessary to ensure that every change in quality remains within the safe variation range of the current bitrate. Thus, $DifIndex$ is compared with $Index_{chg}$. If $DifIndex \geq Index_{chg}$, it means that $RIndex_{best}$ is much higher than $RIndex_{last}$. If we choose $RIndex_{best}$ to follow the current available throughput, an obvious change in quality will be noticed by the user. Thus, we choose $RIndex_{last} + Index_{chg}$ as the bitrate index for the next segment, which provides a relatively rapid response to the available throughput while maintaining a user-friendly quality change. If $DifIndex<Index_{chg}$, then $RIndex_{best}$ can be chosen safely. However, when the buffer level is higher than the maximum buffer threshold and $R_{RIndex_{best}}$ is less than the estimated throughput, a scheme to avoid buffer overflow is required. We increase $RIndex_{best}$ by 1 to prevent buffer overflow. After the application of this overflow-control measure, $DifIndex$ will be less than or equal to $Index_{chg}$, which is still a safe quality change.

When $D_{ori} < 0$, the current available throughput is insufficient to maintain the previous video quality. Bitrate reduction is inevitable. The current buffer level should be carefully considered in this case. If $B_{cur} \leq B_{min}$, then the buffer can be easily emptied if the selected bitrate is higher than the available throughput. We incrementally decrease $RIndex_{best}$ by 1 in a loop until $R_{RIndex_{best}}$ is less than the current available throughput. This strategy will cause the buffer level to increase during the download duration of the next segment to prevent playback freezes, but an unsafe change in quality may occur. If the buffer level satisfies $B_{min} < B_{cur} \leq B_{max}$, the situation is relatively safe and a somewhat conservative scheme is adopted. First, we determine the best possible video quality ($RIndex_{last} - k$) by decreasing bitrate level in a loop to ensure that at least a minimum buffer is preserved. Then, a bitrate level that is no higher than $RIndex_{last} - k$ is

adopted. If $DifIndex$ is less than or equal to $Index_{chg}$, it reveals that the available throughput has not changed considerably. It is unnecessary to switch to $RIndex_{best}$ to adapt to the decreasing throughput. Therefore, $RIndex_{next}$ is set to the minimum value between $RIndex_{best}+1$ and $RIndex_{last}-k$. If $DifIndex$ is larger than $Index_{chg}$, then a large throughput fluctuation has occurred. To adapt to the decreasing throughput quickly and safely, the maximum change in index corresponding to the safe variation range, $Index_{chg}$, is chosen. In accordance with the limits imposed above, $RIndex_{next}$ is set equal to the minimum value between $RIndex_{last}-Index_{chg}$ and $RIndex_{last}-k$. Finally, if $B_{cur}>B_{max}$, then a more conservative scheme is adopted to reduce the buffer. If $DifIndex \le Index_{chg}$, then $RIndex_{next}$ is simply set equal to the last bitrate $RIndex_{last}$ to ensure a smooth video quality. If $DifIndex > Index_{chg}$, then $RIndex_{next}$ is decremented as $RIndex_{last}-1$, just in case an abrupt change in bitrate occurs when the buffer level switches from the current buffer state to the state $B_{min}<B_{cur}\le B_{max}$.

It should be noted that $RIndex_{next}$ must lie in a reasonable range of $[1, m]$ in our algorithm.

4. Experiments and Discussion

In this section, we give an overview of the experimental methodology and evaluate our bitrate adaptation algorithm on our HLS system. After reporting the results of the experiments, we present a simple discussion of our algorithm.

4.1. Experiment Setting

The structure of our HLS system is depicted in Figure 1 and consists of three components, i.e., a HLS server, a HLS client and wired local area network (LAN). On the server side, the original video was encoded in CBR mode to produce 20 available bitrate versions from 100 to 2000 Kbps with a step of 100 Kbps. In addition, each version was chopped into segments of the same duration of 5 seconds (i.e., τ=5). All segments and manifest files were stored on the version 2.4.9 Apache HTTP server that is integrated into Mac Mini 10.10.2. Moreover, the server's Timeout was set to 60 s for alive connections. On the client side, the client was implemented on the Android 4.4.2 platform. All adaptation algorithms were implemented on the client. Three segments with the fourth bitrate index were buffered before the start of video playback. Each subsequent request was sent after the last segment had been completely received. Particularly, when the buffer level would be larger than the target buffer, an idle delay before the sending of the next request was set to account for a limited buffer capacity. In our experiments, the target buffer was set to 7 segment durations. Wired LAN is built by a TP-LINK router and there are no other devices except a HLS server and a HLS client in this LAN.

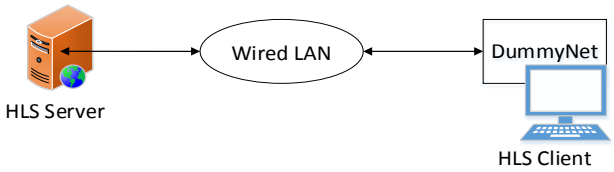


Figure 1. Testbed organization for experiments

For a fair comparison, different algorithms should be evaluated under the same network conditions. The DummyNet network emulator [18] was used to control the available bandwidth. This emulator is easily configured on Windows. To control the available bandwidth on the client side in a simple manner, both DummyNet and our client were installed on a Windows 7

Professional desktop with a 3.10 GHz Intel Core i3 CPU and 4 GB of RAM. In addition, an Android virtual machine was needed to run our client on Windows.

Algorithm 1 Bitrate selection algorithm

Input: $T, T^e, B_{cur}, RIndex_{last}$

Output: $RIndex_{next}$

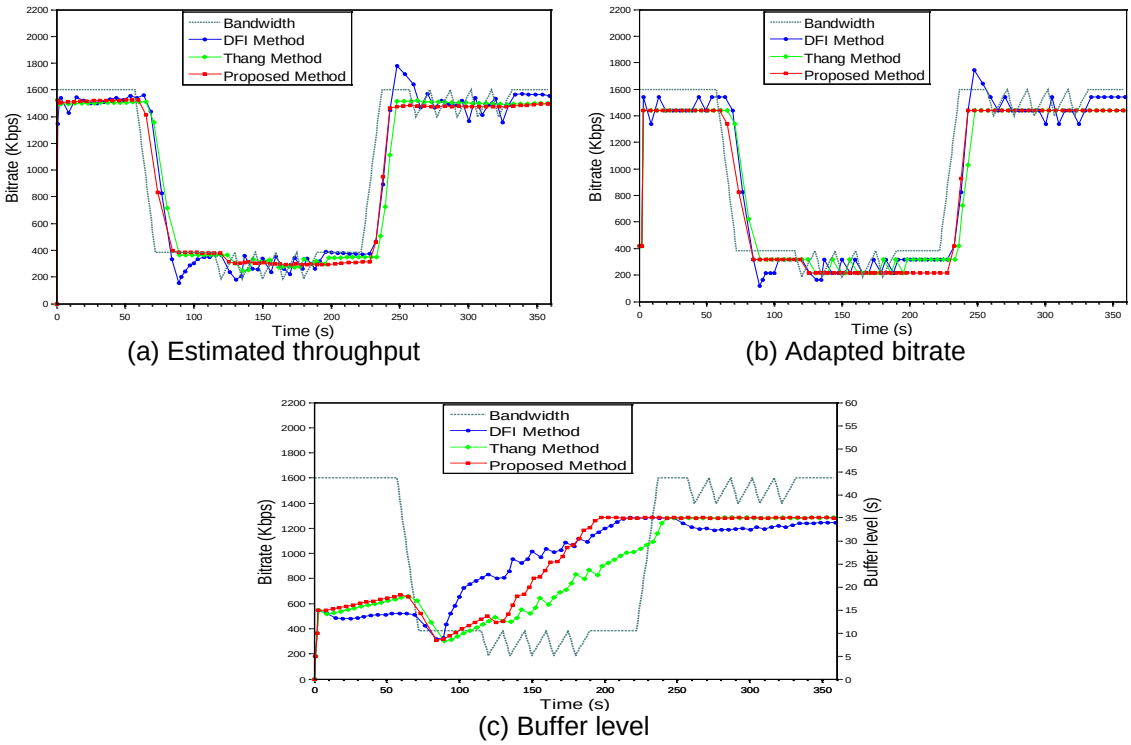
$D_{ori} = RIndex_{best} - RIndex_{last}$
$RIndex_{next} = RIndex_{best}$
if $D_{ori} \geq 0$ **then**
  if $B_{cur} \leq B_{mid}$ **then**
    $RIndex_{next} = RIndex_{last}$
  else
    if $DifIndex \geq Index_{chg}$ **then**
      $RIndex_{next} = RIndex_{last} + Index_{chg}$
    else
      if $B_{cur} \geq B_{max}$ && $R_{RIndex_{best}} < T^e$ **then**
        $RIndex_{next} = RIndex_{best} +$
      end if
    end if
  end if
else
  if $B_{cur} \leq B_{min}$ **then**
    while $R_{RIndex_{best}} > T$ **then**
      $RIndex_{best} = RIndex_{best} -$
    end while
    $RIndex_{next} = RIndex_{best}$
  else if $B_{cur} \leq B_{max}$ **then**
    $k = 0$
    while $(R_{(RIndex_{last}-k)} / T - 1) * \tau > B_{cur} - B_{min}$ **then**
      $k = k + 1$
    end while
    if $DifIndex \leq Index_{chg}$ **then**
      $RIndex_{next} = min(RIndex_{last} - k, RIndex_{best} + 1)$
    else
      $RIndex_{next} = max(RIndex_{last} - k, RIndex_{last} - Index_{chg})$
    end if
  else
    if $DifIndex \leq Index_{chg}$ **then**
      $RIndex_{next} = RIndex_{last}$
    else
      $RIndex_{next} = RIndex_{last} - 1$
    end if
  end if
end if

4.2. Throughput Estimation Experiment

Before our bitrate adaptation algorithm was investigated, our throughput estimation method was evaluated separately in this experiment. The two throughput estimation methods proposed in [7] and [9] were implemented to demonstrate the effectiveness of our method. For simplicity, the methods of [7] and [9] are called the DFI method and the Thang method,

respectively. Complex network conditions with both large fluctuations and short-term fluctuations were emulated using DummyNet.

In our method, in order to obtain a smooth estimation when $\rho < 0.1$ and an aggressive estimation when $\rho > 0.2$, M and ρ_0 were set to 21 and 0.167, respectively. In practice, we regard the fluctuations which are less than one step of bitrates (100 Kbps in our experiments) as the short-term fluctuations. Therefore, T^{nor} was set to 1000 Kbps according to the assumption below equation (6) in section 3.1. B_{min} and B_{max} were set to 1 and 4 segment durations, respectively. The experimental settings for the DFI method and the Thang method were set as described in [7] and [9]. Specifically, in the DFI method, c was set to 0.167 for fairness and ε was set to 0.05 to obtain the optimal results.



(a) Estimated throughput (b) Adapted bitrate

(c) Buffer level

Figure 2. Comparison of different throughput estimation methods

Figure 2(a), Figure 2(b) and Figure 2(c) compare the results in terms of the estimated throughput, adapted bitrate and buffer level, respectively. Note that the bandwidth curve in each of the following figures represents the theoretical capacity of the link (controlled by DummyNet); the estimated throughputs are the results of the different throughput estimation methods; the adapted bitrate is the highest value that is lower than the estimated throughput, shown as equation (12); and the buffer level, which is shown on the right vertical axis in Figure 2(c), represents the current buffer state in seconds.

The following behaviors can be observed from those figures. In the case of large fluctuations (e.g., 60-80 s and 220-240 s), the estimated throughputs of all methods respond quickly as a result of the dynamic control strategies. For this reason, they all have reasonably safe buffers (higher than 8 s). However, in the case of short-term fluctuations (e.g., 115-190 s and 257-332 s), the throughput estimated by the DFI method varies frequently with the fluctuating bandwidth, which causes fluctuations in the adapted bitrates. The Thang method obtains smooth estimates at high bandwidths (e.g., 257-332 s) but fluctuating estimates when the bandwidth is low (e.g., 115-190 s). By contrast, the proposed method obtains smooth estimates for all short-term fluctuations by virtue of the appropriate design of the normalization factor T^{nor}. Therefore, the proposed method achieves the smoothest adapted bitrate. In short,

the proposed method achieves a smooth response to short-term fluctuations and a fast response to large fluctuations.

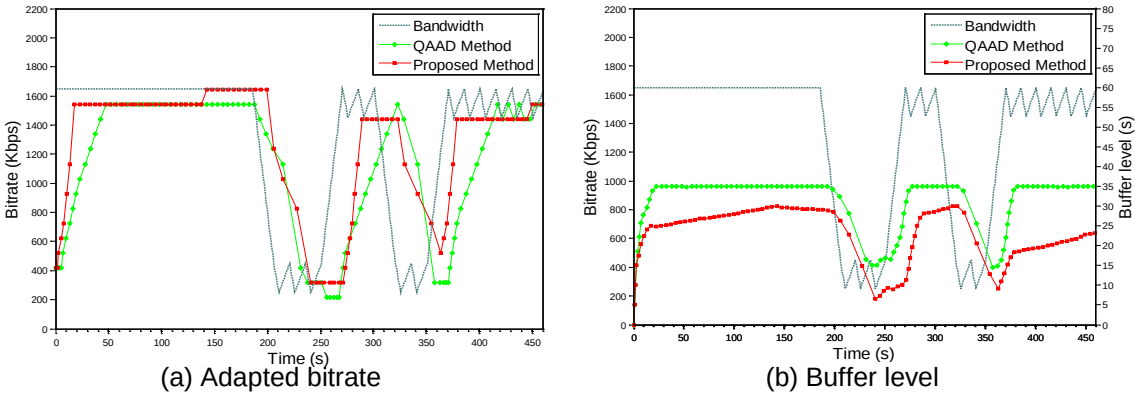
4.3. Bitrate Adaptation Algorithm Experiment

The second experiment was conducted to investigate the performance of our bitrate adaptation algorithm, i.e., the bitrate selection algorithm given the throughput estimation method investigated above. As described above, our bitrate selection algorithm already includes a strategy for preventing buffer underflow. Therefore, to best illustrate the performance of our bitrate selection algorithm, the parameter N was set to 1 for this experiment. The other settings were the same as in the previous experiment. Furthermore, in our bitrate selection algorithm, we set B_{min}, B_{mid} and B_{max} to 1.5, 2 and 6 segment durations, respectively. The bitrate thresholds R_{LowTh}, R_{MidTh} and R_{HigTh} were set to 700, 1000 and 1500 Kbps, respectively.

Another bitrate adaptation algorithm with good performance, QAAD [10], which estimates throughput based on samples of the download throughput, was implemented for comparison. The experimental settings were the same as in [10]. In particular, the predefined marginal buffer length μ was set to 5 segment durations and the minimum buffer length δ was set to 1.5 segment durations for this experiment.

Complex network conditions with both large fluctuations and short-term fluctuations were emulated using DummyNet. The results in terms of the adapted bitrates and buffer levels are compared in Figure 3. The adapted bitrates are the outputs of the bitrate adaptation algorithms.

From Figure 3(a), we can see that when bandwidth is sharply decreasing (e.g., 186-206 s and 301-321 s), the QAAD algorithm result in large changes in bitrate (e.g., 214-231 s and 342-358 s) because it initially attempts to change quality with the smallest possible step in bitrate. By contrast, the proposed algorithm attempts to ensure that each quality change is safe. This measure achieves smoother changes in quality. Moreover, the proposed algorithm has a faster reaction time in reaching the optimal bitrate (e.g., 0-48 s, 266-323 s and 364-419 s). Thus, the proposed algorithm yields a higher adapted bitrate. From Figure 3(b), it is evident that the proposed algorithm achieves a safe buffer.



(a) Adapted bitrate (b) Buffer level

Figure 3. Comparison of different bitrate adaptation algorithms

To more clearly illustrate the advantages of the proposed algorithm, five additional experiments were conducted. We evaluated the performance based on the average values of the five sets of results. Several statistics regarding the tested algorithms are provided in Table 1. The bitrate values are shown in the first three rows in units of Kbps. The next three rows concern the buffer values, and the results in terms of quality changes are shown in the last two rows. The “Maximum change in bitrate” in the third row is the largest bitrate difference between any two consecutive segments. The standard deviation (STD) in the second and fifth rows is used to quantify the amount of variation of the bitrate and buffer. The “Number of unsafe quality

changes" reported in the last row refers to changes in quality that are outside of the safe variation range of the current bitrate.

Table 1. Statistics of Average Values of Five Experiments

Statistics	QAAD	Proposed algorithm	Improvement rate
Average bitrate (Kbps)	1122	1232	9.8%
STD of bitrates (Kb/s)	532	450	15.4%
Maximum change in bitrate (Kbps)	937	408	56.5%
Average buffer level (s)	29.8	21.7	-27.2%
STD of buffer level (s)	7.78	7.01	9.9%
Number of interruptions	0	0	0%
Number of quality changes	52.6	31.4	40.3%
Number of unsafe quality changes	2.6	2	23.1%

From the statistics, we can see that the QAAD algorithm has a higher buffer level because of its conservative strategy and that neither algorithm suffers any interruptions. In fact, a higher buffer with no interruptions does not provide a better QoE for the user. Because the user will not notice a change in the buffer level unless the buffer is exhausted. As expected, the proposed algorithm results in fewer (unsafe) quality changes, smaller maximum change in bitrate and lower STD of bitrates, which implies that our bitrate adaptation algorithm can obtain smoother video quality. Furthermore, the proposed algorithm achieves a higher average bitrate. All of these results confirm that the proposed algorithm achieves a considerable advantage over the QAAD algorithm.

4.4. Discussion

As mentioned above, a throughput estimation method should be stable in the case of short-term fluctuations while also reacting quickly to large fluctuations. The method proposed in [6] achieves smooth estimation for short-term fluctuations but fails to cope with large fluctuations. The DFI method and the Thang method respond quickly to large fluctuations but yield fluctuating estimates in the case of short-term fluctuations. The results of the first experiment show that our throughput estimation method behaves well for both short-term fluctuations and large fluctuations.

In our throughput estimation method, T^{nor} is decided by the step of bitrates. All the steps are equal in our experiments. When the steps are not equal, this situation needs further research; for example, the smallest step can be chosen to decide T^{nor}. Besides, from the observations, we note that the normalization factor T^{nor} plays an important role in the estimation of short-term fluctuations and the parameter M plays an important role in the estimation of large fluctuations. With further research, a dynamic scheme can be designed for determining T^{nor} and M based on the characteristics of different networks.

The algorithm proposed in [19] attempts to maintain a stable bitrate to ensure a smooth bitrate. However, this strategy causes eventual abrupt changes in bitrate in the case of large, rapid decreases in the available throughput. When the available throughput is increasing, the QAAD algorithm changes the quality with the smallest possible step to achieve a smooth bitrate. However, this strategy does not take full advantage of the available bandwidth. The proposed bitrate adaptation algorithm changes the quality within the safe variation range of the current bitrate. This strategy avoids abrupt change in bitrate and gets a high bandwidth utilization. The results of the second experiment indicate that our algorithm can get higher average quality and smoother video quality. It should be noted that the thresholds used to define our piecewise function are suitable for slow-motion [16] videos and some fast-motion videos. Thus, for videos with faster motion, these thresholds should be reset based on the JND.

In most bitrate adaptation algorithms, m different video qualities must inevitably be traversed to select the best bitrate ($R_{RIndex_{best}}$). This is also true of our algorithm. Therefore, our algorithm's time complexity is $O(m)$, identical to that of the QAAD algorithm.

In this paper, we focused purely on improving the performance of our bitrate adaptation algorithm for a single client. However, the case of multiple clients is both more practical and more challenging [20]. The question of how to achieve a balance between efficiency and

fairness of the allocation of throughput resources poses a considerable challenge when multiple clients share a common link. This will be an interesting direction for future research.

5. Conclusion and Future Work

In this paper, a novel bitrate adaptation algorithm which includes a throughput estimation method and a bitrate selection algorithm was proposed for HTTP adaptive streaming. We implemented our throughput estimation method by predicting the difference between the estimated and instantaneous throughputs. Our bitrate selection algorithm was realized based on piecewise functions to determine the variation range of the current bitrate. The proposed method and algorithm were tested on our HLS system over a network emulated using DummyNet. The experimental results show that 1) our throughput estimation method yields a smooth response to short-term fluctuations and a fast response to large fluctuations and 2) our bitrate adaptation (selection) algorithm results in smoother changes in quality with a higher average quality.

We have verified the good performance of our bitrate adaptation algorithm for a single client, and in future research, we will explore its feasibility for multiple clients, considering the efficiency and fairness of the allocation of throughput resources.

Acknowledgements

This work was partly supported by the National 863 Project (No. 2015AA015802) and the Special Fund for Strategic Pilot Technology Chinese Academy of Sciences (No. XDA06040501). The authors would like to thank the anonymous reviewers for their valuable comments.

References

[1] Yanan Z, Xiangyang G, Wendong W, Xirong Q. *A Rate Adaptive Algorithm for HTTP Streaming*. 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems. Hangzhou. 2012: 529-532.
[2] Kamel N, Lanet JL. Analysis of HTTP Protocol Implementation in Smart Card Embedded Web Server. *International Journal of Information & Network Security*. 2013; 2(5): 417-428.
[3] Kupka T, Halvorsen P, Griwodz C. *An Evaluation of Live Adaptive HTTP Segment Streaming Request Strategies*. IEEE Conference on Local Computer Networks. Bonn. 2011: 604-612.
[4] Malinovski T, Trajkovik V, Vasileva M. Students' Perceptions during Integration of Computer Games in Primary Education: QoE Analysis. *International Journal of Informatics and Communication Technology*. 2014; 3(1): 13-22.
[5] Bin L, Qinghua Z, Weizhan Z. *A Rate Adaptation Solution for Distance Education System over HTTP Streaming*. 2013 IEEE International Conference on High Performance Computing and Communications. Zhangjiajie, Hunan, China. 2013: 2385-2389.
[6] Akhshabi S, Begen AC, Dovrolis C. *An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP*. Proceedings of the second annual ACM conference on Multimedia systems. San Jose, CA, United states. 2011: 157-168.
[7] Kim YH, Shin J, Park J. Design and Implementation of a Network-Adaptive Mechanism for HTTP Video Streaming. *ETRI Journal*. 2013; 35(1): 27-34.
[8] Gerla M, Ng BKF, Sanadidi MY, Valla M, et al. TCP Westwood with Adaptive Bandwidth Estimation to Improve Efficiency/Friendliness Tradeoffs. *Computer Communications*. 2004; 27(1): 41-58.
[9] Truong CT, Quang-Dung H, Jung WK, Anh TP. Adaptive Streaming of Audiovisual Content Using MPEG DASH. *IEEE Transactions on Consumer Electronics*. 2012; 58(1): 78-85.
[10] Suh D, Jang I, Pack S. *QoE-Enhanced Adaptation Algorithm over DASH for Multimedia Streaming*. 2014 International Conference on Information Networking. Phuket. 2014: 497-501.
[11] Yuming C, Xiaoquan Y, Jia W, Li S. *A QoE Friendly Rate Adaptation Method for DASH*. Broadband Multimedia Systems and Broadcasting (BMSB). Beijing. 2014: 1-6.
[12] Huang TY, Johari R, McKeown N, Watson M. *A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service*. Proceedings of the 2014 ACM conference on SIGCOMM. Chicago, IL, United states. 2014: 187-198.
[13] Stockhammer T. *Dynamic Adaptive Streaming over HTTP-Standards and Design Principles*. 2nd Annual ACM Multimedia Systems Conference, MMSys'11. San Jose. 2011: 133-143.
[14] Adzic V, Kalva H, Furht B. Optimizing Video Encoding for Adaptive Streaming over HTTP. *IEEE Transactions on Consumer Electronics*. 2012; 58(2): 397-403.

[15] Gouache S, Bichot G, Bsila A, Howson C. *Distributed & Adaptive HTTP Streaming*. 2011 IEEE International Conference on Multimedia and Expo. Barcelona. 2011: 1-6.
[16] Truong CT, Hoc XN, Anh TP, Nam PN. *Perceptual Difference Evaluation of Video Alternatives in Adaptive Streaming*. 2012 Fourth International Conference on Communications and Electronics (ICCE). Hue. 2012: 322-326.
[17] Truong CT, Hung TL, Anh TP, Yong MR. An Evaluation of Bitrate Adaptation Methods for HTTP Live Streaming. *IEEE Journal on Selected Areas in Communications*. 2014; 32(4): 693-705.
[18] Rizzo L. Dummynet: A Simple Approach to the Evaluation of Network Protocols. *ACM SIGCOMM Computer Communication Review*. 1997; 27(1): 31-41.
[19] Miller K, Quacchio E, Gennari G, Wolisz A. *Adaptation Algorithm for Adaptive Streaming over HTTP*. 2012 19th International Packet Video Workshop (PV). Munich. 2012: 173-178.
[20] Jiang J, Sekar V, Zhang H. *Improving Fairness, Efficiency, and Stability in Http-Based Adaptive Video Streaming with Festive*. Proceedings of the 8th international conference on Emerging networking experiments and technologies. Nice, France. 2012: 97-108.