DOI:10.1587/transcom.2016EBP3250

Publicized:2016/11/16

PAPER

Seamless mobile video streaming over HTTP/2 with gradual quality transitions

Hung T. LE[†], **Thang VU**[††], **Nam PHAM NGOC**[††], ***Nonmembers***, **Anh T. PHAM**[†], ***and*** **Truong Cong THANG**[†], ***Members***

SUMMARY HTTP Adaptive Streaming (HAS) has become a popular solution for media delivery over the mobile Internet. However, existing HAS systems are based on the pull-based HTTP/1.1 protocol, leading to high overheads (e.g., in terms of energy, processing, bandwidth) for clients, servers, as well as network nodes. The new HTTP/2 protocol provides a server push feature, which allows the client to receive more than one video segment for each request in order to reduce request-related overheads. In this study, we propose an adaptation method to leverage the push feature of HTTP/2. Our method takes into account not only the request-related overhead but also buffer stability and gradual transitions. The experimental results show that our proposed method performs well under strong throughput variations of mobile networks.

key words: *HTTP adaptive streaming, adaptation, HTTP/2, server push.*

1. Introduction

Over the past few years, HTTP adaptive streaming (HAS) has become the key technology for delivering multimedia over the mobile Internet thanks to the abundance of Web platforms and broadband connections [1], [2]. In HAS, to cope with network fluctuations, a video is encoded at multiple versions (with different video bitrates), each of which is further divided into small segments. A client initiates a streaming session by downloading a metadata file, which contains a description of different versions and segments. After that, an adaptation engine deployed at the client takes responsibility for deciding which segments are requested, based on the metadata and the current status of the terminal/networks. The video is therefore delivered to the client via a sequence of HTTP request-response transactions.

So far, the target delivery protocol in existing HTTP streaming is HTTP/1.1, where every segment is delivered using a request-response pair [2]. Usually, all segments have the same duration, which is typically from two to ten seconds [3]. If the segment duration is short, the client can react quickly to network variations and select the bitrate for a high video quality. However, the use of short segment durations results in a large number of HTTP requests, which in turn causes high overheads (e.g., in terms of energy, processing, bandwidth) for clients, servers, and network nodes [4].

The recently ratified HTTP version 2 (HTTP/2) provides a new feature called “server push", which enables the client to send one request and then receive multiple objects [5]. Hence, short segment durations can be used in HAS systems without increasing the number of HTTP requests (by asking the server to respond to a request with multiple consecutive segments). Recently, the Moving Picture Experts Group (MPEG) has started working on an extension of its HAS standard, where an HTTP/2-based request can specify the number of requested segments [6]. In the literature, the server push feature with a fixed number of pushed segments per request has been investigated for low-delay streaming [7], low request-related overhead [4] and power efficient mobile streaming [8]. Among existing studies, our previous studies in [9], [10] are the first ones that raise the need to adaptively decide the number of pushed/requested segments in each request to balance the requirements of low request-related overhead and buffer stability. In [11], the number of pushed segments is adapted to avoid the “over-push" problem, in which network resources are wasted when a user decides to stop watching a video after checking the first few seconds. However, the main drawback of [11] is that it only considers stable network conditions.

To the best of our knowledge, no previous work has taken into consideration bitrate variations. It should be noted that in mobile networks, as the throughput is fluctuating, the above methods may result in large bitrate reductions and such rate-switching events definitely reduce the quality of service [12]. In this paper, we extend our preliminary work in [10] and additionally consider bitrate variations. Instead of selecting an *adaptation pair* (R, N) of bitrate R and number of pushed segments N for only one next request, we decide adaptation pairs for some future requests. Since the proposed method considers the bitrate trend in the future, it can provide users with gradual bitrate transitions. More specifically, to find the adaptation pairs for the next requests, we examine all possible changes of bitrate R and number N. The optimal option, which balances the requirements of buffer stability, gradual down-switching and low request-related overhead, is then selected. Experiments using variable throughput conditions of mobile networks show that our method can yield gradual bitrate transitions and a low number of requests while keeping the buffer level stable.

The rest of the paper is organized as follows. We provide the background of HAS in Section 2. Section 3 presents the adaptation problem of video streaming over HTTP/2. After that, our proposed method is given in Section 4. We

[††]Thang Vu and Pham Ngoc Nam are with Hanoi University of Science and Technology, Hanoi, Vietnam. Email: nam.phamngoc@hust.edu.vn.

[†]Hung T. Le, Anh T. Pham and Truong Cong Thang are with University of Aizu, Fukushima, Japan. Email: {d8162102, thang}@u-aizu.ac.jp.

present the experimental results and discussions in Section 5. Finally, Section 6 concludes the paper.

2. Background

2.1 Streaming client in HAS

The block diagram of an HTTP client is shown in Fig. 1. Basically, a video content is delivered from the server to the client by a sequence of HTTP/2 request-responses. Depending on the request, the server responds by one or multiple segments with the same bitrate. All segments downloaded from the server are stored in the *Playback buffer* before moving to the *Player*. A key component of the client is *Adaptation engine,* where an adaptation method is deployed to make decisions on which media parts are requested. A good decision for the next request should be based on the current buffer level (observed from the *Playback buffer*), an estimated throughput (provided by the *Throughput estimation* component) and the *Metadata*. While the metadata can be received at the beginning of the streaming session, the client has to estimate the throughput and measure the buffer level during the whole session.

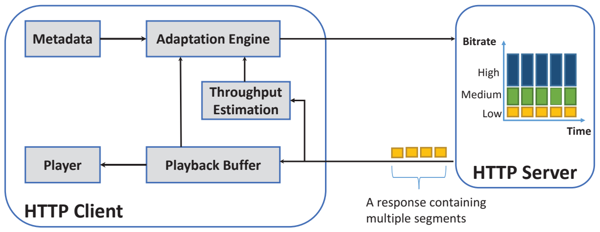


Fig. 1: Block diagram of an HTTP client

In the simplest way, the client can use the instant throughput, which is average throughput $T(l)$ of the last segment l, as the throughput estimate for future segments. However, due to strong throughput variations of mobile networks, the use of the instant throughput may result in short-term fluctuations. A popular solution to cope with this problem is to use the smoothed throughput $T^s(l)$ as follows:

$$T^s(l) = \begin{cases} (1-\delta) \times T^s(l-1) + \delta \times T(l) & \text{if } l > 1, \\ T(1) & \text{if } l = 1, \end{cases} \tag{1}$$

where δ is a weight in the range [0,1]. Hereinafter, the instant throughput and smoothed throughput measured after receiving all segments of request i are referred to as T_i and T_i^s, respectively.

The playback buffer (simply referred to as the buffer) plays an important role in coping with throughput fluctuations in HAS. Basically, to maintain a seamless streaming session, the client should buffer some amount of video data before it can start playing. This period is referred to as the initial delay. After the initial delay, the client switches to the steady stage, where it plays and downloads video segments simultaneously. Note that in this paper, the buffer level, as well as the buffer size, is measured in second. During the steady stage, if the buffer level is large enough, the client can provide users with smooth video quality even under strong throughput variations.

2.2 Quality influence factors

One of the primary goals for HAS is to provide good video quality. Video quality largely depends on the video bitrate as well as rebuffering events [13]. Usually, video rebuffering (caused by buffer underflows) is considered as the worst degradation of video quality, and such events should be avoided by switching the bitrate down [14]. As for video bitrate, an analysis of video quality in [12] showed that continuously switching down to some intermediate bitrate levels provides a better quality than suddenly declining to the target bitrate. However, if the bitrate should be increased, an abrupt increase of bitrate may even improve the perceived quality since users are happy to perceive the quality improvement [15]. Besides, the frequency of the bitrate switches should be minimized for high video quality [13].

Besides video quality, streaming over HTTP also faces the problem of request-related overhead, especially in HTTP/1.1-based systems. The higher number of HTTP requests is, the more energy is consumed at the mobile devices [8]. Moreover, although having small data sizes, these requests significantly increase the processing complexity of network nodes (e.g., proxies and servers) [4]. By applying HTTP/2, it is expected to reduce request-related overheads in HAS.

3. Problem description

In this section, we present the adaptation problem of adaptive streaming over HTTP/2. Some notations and their definitions are shown in Table 1.

3.1 Overview

Suppose that a video is subdivided into small video segments with the same duration of τ seconds. Also, the providers offer the client a set $\mathfrak{R}$ of K bitrate versions $\mathfrak{R} = \{R^k | k = 1, 2, ..., K\}$. The bitrate becomes lower when the version index k decreases.

At certain time t_i right after fully receiving all segments of the current (or last) request i, the client will decide a sequence $\mathbb{S}$ of L adaptation pairs for the next L requests:

$$\mathbb{S} = \left\{ (R_j, N_j) \,\middle|\, j = i+1, i+2, ..., i+L \right\}, \tag{2}$$

where $R_j \in \mathfrak{R}$ and $N_j \geq 1$. Note that the adaptation pair (R_i, N_i) that has already been decided for the last request i is considered as the origin of sequence $\mathbb{S}$.

We assume that the client has a set $\aleph$ of M push strategies $\aleph = \{1, 2, ..., M\}$. Hence, there are totally $M \times K$ options

Table 1: Notations and definitions used in this paper.

Notation	Unit	Definition
τ	second	the media segment duration.
t_i	second	the time instance right after the client completely receives all segments of request i.
T_i	kbps	the instant throughput obtained at time t_i.
T_i^s	kbps	the smoothed throughput obtained at the time t_i.
T_i^e	kbps	the estimated throughput at time t_i for the future segments.
K		the number of available video versions.
$\Re$		the set of available bitrates $\Re = \{R^1, R^2, ..., R^K\}$.
M		the number of push strategies.
$\aleph$		the set of push strategies $\aleph = \{1, 2, ..., M\}$.
μ		the safety margin, used to decide the bitrate in (8) and (16).
(R_j, N_j)		the adaptation pair decided for request j, it contains bitrate R_j and number of pushed segments N_j.
B_i	second	the measured buffer level at time t_i.
B_j^e	second	the estimated buffer level after the client fully receives all segment of request j $(j > i)$.
B_{tar}	second	the buffer target level, it is also the buffer size.
B_{min}	second	the minimum buffer threshold in the rate decrease case.
$\mathbb{S}$		the sequence of adaptation pairs for some future segments.
L		the number of adaptation pairs in sequence $\mathbb{S}$.
$\Im$		the set of sequence candidates in the rate decrease case.

for each adaptation pair (R_j, N_j).

3.2 Buffer Level Estimation and Adaptation Problem

An important objective of adaptation methods is to provide seamless streaming. Therefore, sequence $\mathbb{S}$ should be decided so that the buffer level is not depleted during the session. For this purpose, we estimate the buffer level in the near future, given the current buffer level B_i and estimated throughput T_i^e. Note that the estimated throughput T_i^e could be either instant throughput T_i or smoothed throughput T_i^s depending on the context. The detailed selections are provided later in Section 4.

Let us denote B_j^e the buffer level measured right after the client fully receives all segments of request j $(i < j \le i + L)$. We compute buffer level B_j^e from the previous buffer level B_{j-1}^e as follows. With new N_j segments, the buffer level increases $N_j \times \tau$ seconds. However, the client also spends $N_j \times \frac{\tau \times R_j}{T_i^e}$ seconds on downloading these segments. Therefore, the estimated buffer level B_j^e is computed by

$$B_j^e = \begin{cases} B_{j-1}^e + \Delta_j & \text{if } j > i + 1, \\ B_i + \Delta_j & \text{if } j = i + 1, \end{cases} \tag{3}$$

where Δ_j is the change of buffer level given request j:

$$\Delta_j = N_j \times \tau \times \left(1 - \frac{R_j}{T_i^e}\right). \tag{4}$$

So, to avoid buffer underflows, we have a buffer constraint for selecting sequence $\mathbb{S}$ as follows:

$$B_j^e > B_{min},\ i < j \le i + L, \tag{5}$$

where B_{min} is a predefined buffer threshold.

All sequence options of sequence $\mathbb{S}$ that satisfy condition (5) are gathered into a set $\Im$ of sequence candidates. In the following section, we present our solution to select the optimal sequence from set $\Im$.

4. Proposed adaptation method

This part presents our method in details. The proposed method includes the following distinguishing features:

- Since the method estimates the trend of buffer level in the near future and selects a sequence from set $\Im$, it can avoid buffer underflows even under the strong variations of connection throughput.
- When the throughput drops, the method considers the requirement of quality smoothness and balances it with the requirements of buffer stability and low request-related overhead.
- When the throughput is increased, as the quality and the buffer level are not negatively affected, the method fast increases the bitrate to quickly improve video quality. Additionally, the highest possible number of pushed segments is decided to reduce request-related overheads.

4.1 General adaptation process

Based on the relation between the current bitrate R_i and the instant throughput T_i, we divide our method into two cases: Rate decrease (when $R_i > T_i$) and Rate increase (when $R_i \le T_i$). In both cases, our basic idea to decide sequence $\mathbb{S}$ is to balance the requirements of buffer stability, gradual down-switching and low request-related overhead. The detailed algorithms for the two sub-cases are presented in Subsections 4.2 and 4.3.

Assume that the request sequence is decided at time t_i. In an ideal condition, the client simply sends the requests based on the current (decided) sequence $\mathbb{S}$ until the final request $i + L$. However, at certain time, if the buffer level is reduced below threshold B_{min}, the current sequence $\mathbb{S}$ is aborted immediately, the client then selects pair (R^1, M) for the next request to quickly improve the buffer level.

In addition, during the interval of the current sequence $\mathbb{S}$, if the throughput changes, leading to a significant buffer variation, it is also reasonable to redetermine sequence $\mathbb{S}$. In our method, we decide a new sequence at time t_{j^*} $(j^* > i)$ if the mismatch between the estimated buffer $B_{j^*}^e$ and the actually-measured buffer level B_{j^*} at that time exceeds one segment duration, i.e.

Algorithm 1: Request selection at time t_j

```
Input  : Current sequence $\mathbb{S}$ decided at time $t_i (i < j)$, current
         buffer level $B_j$.
Output: Adaptation pair $(R_{j+1}, N_{j+1})$ for request $j + 1$.
1  begin
2      if $B_j > B_{min}$ then
3          if the client has requested the final adaptation pair of
             sequence $\mathbb{S}$ or $\left|B^e_j - B_j\right| > \tau$ then
4              $i \leftarrow j$;
5              Select new sequence $\mathbb{S}$;
6          else
7              Continue using current sequence $\mathbb{S}$;
8          end
9          Decide pair $(R_{j+1}, N_{j+1})$ of $\mathbb{S}$ for request $j + 1$;
10     else
11         Abort current sequence $\mathbb{S}$;
12         Decide pair $(R^1, M)$ for request $j + 1$;
13     end
14 end
```

$$\left|B^e_{j^*} - B_{j^*}\right| > \tau. \tag{6}$$

Note that the sequence could be redetermined by either the rate decrease case or the rate increase case, depending on the relation between the bitrate and the throughput at time t_{j^*}.

In summary, the general process for pair selection at time t_j, given the current sequence $\mathbb{S}$ (decided at the previous time t_i), is provided in Algorithm 1. Note that this process is invoked at every request by the client.

4.2 Rate decrease case ($R_i > T_i$)

In order to quickly capture throughput behavior in the rate decrease case, we use the instant throughput T_i in estimating the throughput:

$$T^e_i = T_i. \tag{7}$$

As the throughput is higher than the current bitrate, the final bitrate R_{i+L} should be lower than the estimated throughput T^e_i. Usually, a safety margin μ in the range [0, 1] is used to compute the final bitrate R_{i+L}:

$$R_{i+L} = \max\left\{R \mid R \in \mathfrak{R} \wedge R < (1-\mu) \times T^e_i\right\}. \tag{8}$$

To find an optimal sequence $\mathbb{S}$, we introduce a cost function, which is the tradeoff among request-related cost C_{req}, bitrate smoothness-related cost C_{smt}, and buffer-related cost C_{buf}:

$$C = \alpha \times C_{req} + \beta \times C_{smt} + \gamma \times C_{buf}, \tag{9}$$

where α, β and γ are tradeoff parameters.

Since the request-related overhead is inversely proportional to the number of pushed segments, we propose a request cost function based on the average value of the requested segments throughout a sequence:

$$C_{req} = \frac{1}{\frac{1}{L} \times \sum_{j=i+1}^{i+L} N_j}. \tag{10}$$

The smoothness cost is a function with regards to quality changes along the sequence, it is defined by:

$$C_{smt} = \max\left\{V(R_{j-1}) - V(R_j) \mid i < j \le i + L\right\}, \tag{11}$$

where $V(x)$ indicates the version index of bitrate x. It is noted that C_{smt} is the minimum when the changes of bitrate along the sequence are equal.

The buffer cost should depend on the mismatch between the target buffer level B_{tar} and the estimated buffer level B^e_{i+L} (i.e., the buffer level after using sequence $\mathbb{S}$). We propose an exponential function to compute the buffer cost C_{buf}:

$$C_{buf} = \exp\left(B_{tar} - B^e_{i+L}\right), \tag{12}$$

where B^e_{i+L} is computed following (3).

So, the client computes the overall cost C for each sequence candidate $\mathbb{S}$ in set $\mathfrak{I}$ and then selects the optimal sequence that has the lowest cost. Due to strong fluctuations of mobile networks, it is reasonable to set the sequence length L small. In practical implementation in Java, we found that with typical parameters (e.g., $L = 3, M = 4$, and a set of 17 bitrate versions as in [16]), the execution time for deciding sequence $\mathbb{S}$ is just a few milliseconds and thus does not negatively affect the processing time for adaptation.

4.3 Rate increase case ($R_i \le T_i$)

In this case, the smoothness of quality is also not vital. Therefore, it is reasonable to decide an adaptation pair only for the next request $i+1$ (i.e., the sequence length $L = 1$). Our idea is to maintain the current bitrate until the buffer level reaches the target buffer level B_{tar}. The detailed adaptation logic is as follows.

To avoid short-term throughput fluctuations, we additionally use the smoothed throughput T^s_i in estimating the throughput:

$$T^e_i = \min\left\{T^s_i, T_i\right\}. \tag{13}$$

Here, T^s_i is computed by (1) with $\delta = 0.125$, which is recommended for smoothing computation [17].

We now divide the rate increase case into two sub-cases depending on the current buffer level B_i. If $B_i < B_{tar}$ (i.e., the first sub-case), the current bitrate R_i is maintained until the buffer level reaches the target buffer level B_{tar}:

$$R_{i+1} = R_i. \tag{14}$$

The number of pushed segments N_{i+1} is computed by:

$$N_{i+1} = \min\left\{M, \min\left\{N | B^e_{i+1} \ge B_{tar}\right\}\right\}, \tag{15}$$

where B^e_{i+1} is obtained from (3).

In the second sub-case ($B_i \ge B_{tar}$), the client immediately switches up to the highest bitrate, which is lower than throughput estimate T^e_i:

$$R_{i+1} = \max\left\{R \mid R \in \mathfrak{R} \wedge R < (1-\mu) \times T^e_i\right\}. \tag{16}$$

Algorithm 2: Sequence selection at time t_i

```
Input  : Instant throughput $T_i$, smoothed throughput $T_i^s$, current
         buffer level $B_i$.
Output: Sequence $\mathbb{S}$.
1  if $R_i > T_i$ then
2      // Rate decrease case
3      $T_i^e \leftarrow T_i$;
4      for each sequence candidate $\mathbb{S}$ in set $\mathfrak{I}$ do
5          Compute a set of corresponding buffer levels
           $\{B_j^e | i < j \leq i + L\}$;
6          Compute request cost $C$ following (9);
7      end
8      Select the optimal sequence $\mathbb{S}$ which has the lowest cost;
9  else
10     // Rate increase case
11     $T_i^e \leftarrow \min\{T_i^s, T_i\}$;
12     if $B_i < B_{tar}$ then
13         Compute $(R_{i+1}, N_{i+1})$ following (14) and (15);
14     else
15         Compute $(R_{i+1}, N_{i+1})$ following (16) and (17);
16     end
17     $\mathbb{S} \leftarrow \{R_{i+1}, N_{i+1}\}$;
18 end
```

The number of pushed segments N_{i+1} is set to M in order to minimize request-related overheads:

$$N_{i+1} = M. \tag{17}$$

It should be noted that the existing methods strictly decide the number of pushed segment in advance (e.g., [7]) or based on buffer level only (e.g., [10]). Meanwhile, our method flexibly makes decisions based on both buffer level and connection throughput. Our general procedure in both rate decrease and rate increase cases is summarized in Algorithm 2.

5. Experiments and Discussions

5.1 Experiment settings

In this section, we evaluate our method using a simple bandwidth trace and a complex bandwidth trace obtained from a mobile network. For reference methods, we use the Push-N method used in [4], [7] and the request-buffer-based (RBB) method proposed in [10]. In the former method, N segments are sent for each request. Meanwhile, in the latter method, the number of pushed segments is adaptively decided to balance buffer stability and low request-related overhead. Note that these two methods decide the bitrate based on the estimated throughput only. Here, our focus is on the adaptation behavior in the steady stage. In the initial buffering stage, the simple Push-N method with N = 1 is employed.

The testbed of our experiments is similar to that of [10], which includes an HTTP/2 web server, an HTTP/2 client and an IP network. The IP network includes a router and wired connections connecting the server and the client. On the server side, an HTTP/2-enable server is installed in Ubuntu 14.04 LTS. A test video is encoded at 17 bitrate versions based a DASH dataset in [16], i.e. $\mathfrak{R}$ = {100, 150, 200, 250, 300, 400, 500, 700, 900, 1200, 1500, 2000, 2500, 3000, 4000, 5000, 6000} (kbps). All segments have the same segment duration, which is provided later in each experiment. The client is implemented in Java and runs on a Windows 7 notebook with 2.4 GHz core i5 CPU. Dummynet tool [18], a network emulator, is installed at the client to emulate the characteristics of mobile networks. The round trip time (RTT) is set to 100ms. The packet loss rate is set to 0%, assuming that the packet loss are already included in the fluctuations of employed bandwidth traces.

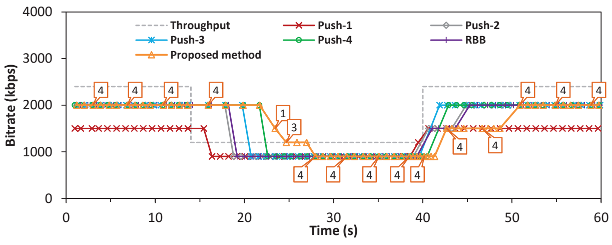
The three evaluated methods are employed with the same target buffer level (or the buffer size) B_{tar} = 15s and the same safety margin $\mu = 0.05$. We implement the Push-N method with 4 options N = 1, 2, 3 and 4. The set of push strategies of the two other methods is set to $\aleph$ = {1, 2, 3, 4}. The RBB method is employed with tradeoff parameter $\alpha = 0.2$. Our method is implemented with B_{min} = 3s, i.e. 3 segment durations with option τ = 1s.

In our method, a streaming provider can adjust the balance between the requirements for low request-related overhead, gradual transitions, and buffer stability by changing the tradeoff parameters α, β and γ. Basically, we fix α and select two others. Given $\alpha = 10$, the contribution of the request-related component to the overall cost C is at most 10 (i.e. when the client decides only one segment per request). Since we want to prioritize the requirement of gradual transitions, β is selected so that the contribution of the smoothness-related cost is higher than that of the request-related cost. With parameter γ, because the buffer-related cost C_{buf} varies in a wide range from 2^0 to $2^{B_{tar}}$, parameter γ should be small to reduce the contribution of the buffer-related cost to the overall cost C. Based on our experience, good empirical values of the parameters α, β, and γ are 10, 13.5, and 0.08, respectively. In our future work, the tradeoff of these factors and other additional factors (e.g. minimum bitrate, buffer level variation, etc.) will be investigated.

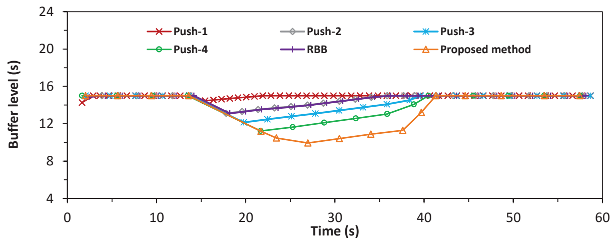
5.2 Simple bandwidth scenario

As mentioned earlier, one of the main challenges in HAS is to provide gradual down-switching when the throughput suddenly drops. Thus, we compare the methods when the connection throughput suddenly declines from 2400kbps to 1200kbps as in Fig. 2a. In this experiment, the segment duration is set to 1 second, which is lower than that of popular HTTP streaming systems (i.e. typically from 2 to 10 seconds [3]). Fig. 2 shows the adaptation results of the three methods.

The Push-N and the RBB methods aggressively switch the bitrate according to throughput variations. As a consequence, they all result in a large bitrate change. Especially, the Push-1 option has the smallest bitrate curve since this option does not leverage the push feature of HTTP/2. From Fig. 2a, we also observe that the time to change the bitrate of the RBB and Push-N methods are dependent on the number of pushed segments, which is decided around time t = 13s (before the throughput drops). As seen in Fig. 2, the lower number of pushed segments, the earlier the bitrate is changed

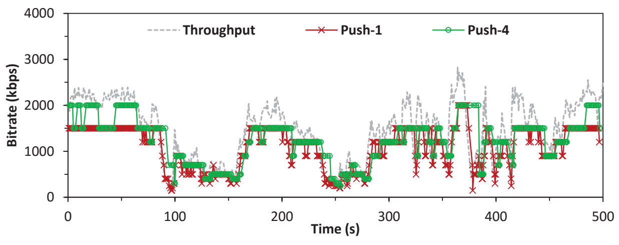


(a) Adapted bitrate

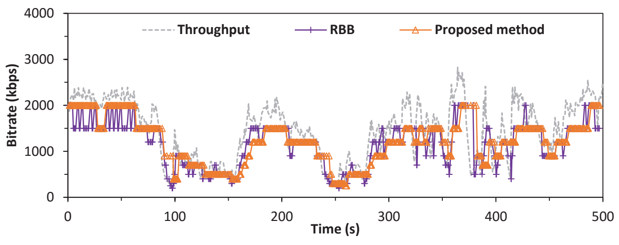


(b) Resulting buffer level

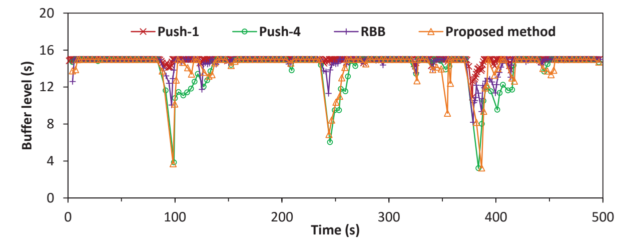
Fig. 2: Adaptation results of the three methods in simple bandwidth scenario.



(a) Adapted bitrate of the push-N method with N = 1 and 4



(b) Adapted bitrate of the RBB and proposed methods



(c) Resulting buffer level

Fig. 3: Adaptation results of the three methods in complex bandwidth scenario.

and so, the more stable the buffer level becomes.

As for our proposed method, besides the number of pushed segments and buffer level, it also considers bitrate variations. Fig. 2a shows that the proposed method gradually changes the bitrate from 2000kbps to 900kbps. Specifically, at time t = 22s after detecting a throughput drop, the method decides the sequence $\mathbb{S} = \left\{(R_j, N_j)\right\}$ = {(1500kbps, 1); (1200kbps, 3); (900kbps, 4)} for the next 3 requests. It is seen that in our method, both bitrate and number of pushed segments are adaptively decided. The results show that our method requires 3 requests for downloading the next 8 segments (i.e., around 3 segments per request), provides gradual down-switching, and has good buffer levels (i.e. higher than 9s). That means, when the throughput is reduced, our method can balance between gradual down-switching, buffer stability and number of requests.

5.3 Complex bandwidth scenario

This part investigates the performance of the three methods using a time-varying bandwidth trace obtained from a mobile network [19]. As shown in Fig. 3a, the connection throughput widely varies from 300kbps to 2800kbps. All other settings are the same as before.

Fig. 3 shows the adaptation results of the three methods. Because the Push-2 and Push-3 options have similar performances to that of the Push-4 option, they are not included in Fig. 3 for the sake of clarity, but they are still shown in our later statistics. As seen in Figs. 3a and 3b, the bitrate curves provided by both Push-N and RBB methods widely vary according to throughput fluctuations. Meanwhile, our proposed method provides bitrate stability and gradual down-switching. For example, during 60s ∼ 90s, our method maintains a bitrate of 1500kbps even the throughput is fluctuating. At time t = 375s, when the throughput suddenly drops, it changes the bitrate gradually from 2000kbps to 700kbps while other methods directly jump to a bitrate around 400kbps. Fig. 3c shows that the Push-N method with option N = 1 has the most stable buffer level curve while our method's buffer level curve is also good. The minimum buffer level of our method (3.2s) is still higher than the predefined threshold B_{min}.

Some statistics of the adaptation results are provided in Table 2. The statistics are related to adapted bitrate, buffer level, number of requests and version decreases. Note that version decreases are used instead of bitrate decreases because a change of 1000kbps at a high bitrate level may be not as severe as a change of 500kbps at a low bitrate level.

As expected, the Push-1 option has the lowest average bitrate (1081kbps) as the client has to wait one RTT before starting receiving every segment. The statistics of buffer level demonstrate that all adaptation methods can avoid buffer underflows. Although giving the highest minimum buffer level (10.9s), the Push-1 option results in the highest number of requests (500) and thus a high request-related overhead. Meanwhile, the RBB method dynamically decides the number of pushed segments based on the buffer level. The adaptation results show that the RBB method

Table 2: Statistics of adaptation results in complex bandwidth scenario.

Statistics	Push-N				RBB	Proposed method
	N=1	N=2	N=3	N=4		
Average bitrate (kbps)	1081	1148	1162	1184	1152	1180
Minimum buffer level (seconds)	10.9	8.9	6.2	3.2	8.2	3.2
Number of requests	500	250	167	125	252	131
Number of version decreases	74	62	38	32	60	21
Average version decrease	1.3	1.3	1.5	1.4	1.4	1.3
Maximum version decrease	6	5	5	5	5	3

Table 3: Statistics of adaptation results in complex bandwidth scenario with a segment duration of 500ms.

Statistics	Push-N				RBB	Proposed method
	N=1	N=2	N=3	N=4		
Average bitrate (kbps)	1039	1119	1138	1164	1132	1218
Minimum buffer level (s)	12.3	10.9	9.9	8.6	10.7	8.1
Number of requests	1000	500	334	250	371	264
Number of version decreases	155	87	68	54	73	48
Average version decrease	1.1	1.2	1.3	1.3	1.3	1.0
Maximum version decrease	6	6	5	5	6	2

can balance between buffer stability and low request-related overhead. From the third row of Table 2, we see that our method requires only 131 requests during a whole session, which is comparable to 125 requests of the Push-4 option. This good result is because our method decides the number of pushed segments based on not only the buffer level but also the throughput trend (increase or decrease).

In terms of version decreases, the Push-1 option has the highest number of version decreases (up to 74) while that of our method is the lowest (21). Our method's result is even 30 percent lower than that of the second (32). Moreover, our proposed method provides a small average value (1.3) and the smallest maximum value (3) of version decreases. That means, our method reasonably switches the bitrate down to avoid large version decreases. The above statistics confirm that our proposed method can provide a high average bitrate, buffer stability, a low number of requests as well as gradual transitions when the throughput is reduced.

As already mentioned, HTTP/2-based streaming supports small segment durations. Therefore, we investigate the performances of the three methods with a smaller segment duration. Table 3 shows the statistics for the case that the segment duration is reduced to 500ms. It is seen that all the methods have better buffer behaviors but higher numbers of requests in comparison with those for the case of 1-second segment duration. Among the three methods, our method has the highest average bitrate (1218kbps). In addition, the proposed method also provides the smoothest down-switching (with only 48 version decreases and at most 2 version decreases per switch) while still providing a very low number of requests (264).

5.4 Discussions

From the above experiments, it can be seen that although the Push-N method can yield buffer stability (when N = 1) or low request-related overhead (when N = 4), setting the value of N in the varying conditions of connection throughput is not easy. The RBB method can decide the number of pushed segment for each request to balance the two mentioned factors. However, the main disadvantage of these methods is that they decide the bitrate regardless of bitrate switching and thus leading to quality instability and sudden quality decreases.

To overcome this problem, the proposed method further considers the requirements of gradual down-switching in making decisions. For gradual transitions when switching down the bitrate, our method decides the adaptation pair (R, N) not only for one next request but also for some future requests. The adaptation rule to provide the gradual down-switching, together with the policy of immediate up-switching when the buffer is full, helps our method to have a high average bitrate during a whole session.

Interestingly, although deciding the sequence $\mathbb{S}$ of adaptation pairs for some future segments, our method is not sacrificed by the buffer stability since it considers the buffer trend in the future. During a session, the buffer level is measured after receiving every response; if the estimated buffer levels become inaccurate, our method can redetermine sequence $\mathbb{S}$ to adapt to new conditions of the client/networks. This feature enables our method to take advantage of the buffer to better cope with throughput fluctuations.

In our method, the number of request over a streaming session is reduced since the method decides the number of requests depending on both buffer level and throughput trend. If the throughput is reduced, the number of pushed segments for each request is adaptively decided. Otherwise, the client can select the highest option since the buffer and the video quality are not negatively affected. It is different from the other methods where the number of request is set in advance [4], [7] or is based on buffer level only [10].

6. Conclusions

The recently ratified HTTP/2 protocol enables video streaming systems to provide users with higher flexibility compared to traditional HTTP/1.1-based systems. This paper has presented a novel adaptation method with gradual down-switching for video streaming over HTTP/2. In our method, the client considers adaptation for some future segments rather than just the next segment. If the bitrate should be reduced, the client selects the optimal selection to balance the requirements of buffer stability, low request-related overhead and gradual transitions. If the bitrate is to be increased, the method rapidly increases the bitrate to quickly improve video quality. Experiments show that our proposed method can provide users with high average bitrates, buffer stability and low numbers of requests while outperforming the reference methods in terms of quality switching.

Acknowledgment

The authors sincerely thank the anonymous reviewers for the constructive comments. This work was partially supported by JSPS KAKENHI Grant Number 15K00134 and the competitive funds of the University of Aizu.

References

[1] T. Stockhammer, "Dynamic adaptive streaming over http: Standards and design principles," Proc. Second Annual ACM Conference on Multimedia Systems (MMSys '11), pp.133–144, 2011.

[2] 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, Feb. 2012.

[3] T.C. Thang, H.T. Le, 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, Dec. 2013.

[4] S. Wei and V. Swaminathan, "Cost effective video streaming using server push over http 2.0," Proc. 16th International Workshop on Multimedia Signal Processing (MMSP2014), pp.1–5, Sep. 2014.

[5] M. Belshe, R. Peon, and M. Thomson, "Hyper transfer protocol version 2 (http/2)," RFC 7540, May 2015.

[6] Working draft for ISO/IEC 23009-6, "Dash over full duplex http-compatible protocols (fdh)," Oct. 2015.

[7] S. Wei and V. Swaminathan, "Low latency live video streaming over http 2.0," Proc. 24th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '14), pp.37–42, Mar. 2014.

[8] S. Wei, V. Swaminathan, and M. Xiao, "Power efficient mobile video streaming using http/2 server push," Proc. 17th International Workshop on Multimedia Signal Processing (MMSP '15), pp.1–6, Oct. 2015.

[9] D.V. Nguyen, H.T. Le, P.N. Nam, A.T. Pham, and T.C. Thang, "Request adaptation for adaptive streaming over http/2," Proc. of the IEEE International Conference on Consumer Electronics (ICCE 2016), pp.189–191, Jan. 2016.

[10] D.V. Nguyen, H.T. Le, P.N. Nam, A.T. Pham, and T.C. Thang, "Adaptation method for video streaming over http/2," IEICE Communications Express, vol.5, no.3, pp.69–73, 2016.

[11] M. Xiao, V. Swaminathan, S. Wei, and S Chen, "Evaluating and improving push based video streaming with http/2," Proc. 26th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '16), May 2016.

[12] R.K.P. Mok, X. Luo, E.W.W. Chan, and R.K.C. Chang, "Qdash: A qoe-aware dash system," Proc. of the 3rd Multimedia Systems Conference (MMSys '12), pp.11–22, Feb. 2012.

[13] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia, "A survey on quality of experience of http adaptive streaming," IEEE Communications Surveys Tutorials, vol.17, no.1, pp.469–492, 2015.

[14] T. Hoßfeld, D. Strohmeier, A. Raake, and R. Schatz, "Pippi longstocking calculus for temporal stimuli pattern on youtube qoe: 1+1=3 and 1·4≠4·1," Proc. of the 5th Workshop on Mobile Video (MoVid '13), pp.37–42, Feb. 2013.

[15] M. Grafl and C. Timmerer, "Representation switch smoothing for adaptive http streaming,," Proc. 4th International Workshop on Perceptual Quality of Systems (PQS 2013), pp.178–183, Sep. 2013.

[16] S. Lederer, C. Müller, and C. Timmerer, "Dynamic adaptive streaming over http dataset," Proc. 3rd Multimedia Systems Conference (MMSys '12), pp.89–94, Feb. 2012.

[17] V. Paxson, M. Allman, H.J. Chu, and M. Sargent, "Computing tcp's retransmission timer," 2011.

[18] L. Rizzo, "Dummynet: A simple approach to the evaluation of network protocols," SIGCOMM Comput. Commun. Rev., vol.27, no.1, pp.31–41, Jan. 1997.

[19] C. Müller, S. Lederer, and C. Timmerer, "An evaluation of dynamic adaptive streaming over http in vehicular environments," Proc. 4th Workshop on Mobile Video (MoVid '12), pp.37–42, 2012.

Hung T. Le received the B.E. degree from Hanoi University of Technology (Vietnam) and M.Sc degree in University of Aizu (Japan) in 2012 and 2014, respectively. He is currently working toward a Ph.D. degree in Computer Science and Engineering at University of Aizu. Hung's study in Japan is funded by a Japanese government scholarship (MonbuKagaku-sho). His major research interests are multimedia networking, video adaptation, QoE support.

Thang Vu received the B.E. degree from Hanoi University of Science and Technology in 2016. He is currently a research assistant in the ESRC Lab., Hanoi University of Science and Technology. His research interests include Quality of Experience (QoE), multimedia networking, and content adaptation.

Nam Pham Ngoc received B.E. degree in Electronics and Telecom. from Hanoi University of Science and Technology (Vietnam) and M.Sc. degree in Artificial Intelligence from K.U. Leuven (Belgium) in 1997 and 1999, respectively. He was awarded a Ph.D. degree in Electrical Engineering from K.U. Leuven in 2004. From 2004 until now he has been working at Hanoi University of Science and Technology, Vietnam. His research interests include QoS management at end-systems for multimedia applications, reconfigurable embedded systems and low-power embedded system design.

Anh T. Pham Anh T. Pham received the B.E. and M.E. degrees, both in Electronics Engineering from the Hanoi University of Technology, Vietnam in 1997 and 2000, respectively, and the Ph.D. degree in Information and Mathematical Sciences from Saitama University, Japan in 2005. From 1998 to 2002, he was with the NTT Corp. in Vietnam. Since April 2005, he has been on the faculty at the University of Aizu, where he is currently Professor and Head of Computer Communications Laboratory with the Division of Computer Engineering. Professor Pham's research interests are in the broad areas of communication theory and networking with a particular emphasis on modeling, design and performance evaluation of wired/wireless communication systems and networks. He has authored/co-authored more than 140 peer-reviewed papers, including 40+ journal articles, on these topics. Professor Pham is senior member of IEEE. He is also member of IEICE and OSA.

Truong Cong Thang received the B.E. degree from Hanoi University of Science and Technology, Vietnam, in 1997 and the Ph.D. degree from KAIST, Korea, in 2006. From 1997 to 2000, he worked as a network engineer in Vietnam Post & Telecom (VNPT). From 2007 to 2011, he was a Member of Research Staff at Electronics and Telecommunications Research Institute (ETRI), Korea. He was also an active member of Korean and Japanese delegations to standard meetings of ISO/IEC and ITU-T from 2002 to 2014. Since 2011, he has been an Associate Professor of University of Aizu, Japan. His research interests include multimedia networking, image/video processing, content adaptation, IPTV, and MPEG/ITU standards.