Adaptation Method for Streaming of VBR Video Over HTTP/2

Nguyen Thi Kim Thoa, Nguyen Minh, Nguyen Hai Dang, Pham Hong Thinh, Pham Ngoc Nam
Hanoi University of Science and Technology, No. 1, Dai Co Viet, Hai Ba Trung, Hanoi, Viet Nam.
Received: March 27, 2017; accepted: June 9, 2017

Abstract

Recently, Dynamic Adaptive Streaming over HTTP (DASH) has become popular for video delivery in multimedia network. However, HTTP streaming is currently based on the pull-based HTTP/1.1 protocol which requires a large number of client requests for each streaming session. Moreover, most of adaptive streaming methods have just focused on the case of CBR (constant bitrate) video. In this paper, we introduce a new method for quality adaptation of VBR (variable bitrate) videos in on-demand streaming over the new HTTP/2 protocol using server push feature. To the best of our knowledge, this is the first study on streaming VBR videos over HTTP/2. Experimental results show that the proposed method can provide a lower number of requests, a higher average quality and more smooth video quality than existing methods.
Keywords: Adaptive Streaming, HTTP/2, Server Push.

1. Introduction

Recently, multi-bitrate adaptive streaming such as HTTP adaptive streaming (HAS) has become the de facto standard for over-the-top video streaming [1]. In HAS, video content is encoded at multiple quality levels and temporally divided into short segments. The client can select the quality level for every video segment based on the network and terminal situation. Currently, HTTP1/1 is the common delivery protocol in Dynamic Adaptive Streaming over HTTP (DASH) with the main operation mechanism is request-respond. Specifically, after downloading every segment, the client sends a request to choose a suitable version for the next one. The server then responses a segment with the corresponding version to the client. Normally, video segments are set to fixed-duration from 2s to 10s [2]. Clearly, the benefit of long segment duration is fewer requests and less overhead, leading to higher overall throughput. Nevertheless, the client could only adapt to the network change when it receives the whole video segment, causing slow response rate and, as a result, buffer instability. Moreover, long segment duration results in large delay [3]. A straightforward solution is to use short segment durations, which certainly imposes an explosion in number of requests that generates significant overheads. This increases the processing complexity of network nodes and reduces the overall throughput [4].

Recently, a new version of HTTP protocol is proposed, called HTTP/2, intended as a higher performance alternative to HTTP/1.1 [5]. It introduces a dominant feature called *server push*. The use of this feature enables server to push multiple consecutive segments with the same version per client's request. Therefore, short segment duration could be used without requiring too many requests.

Using the server push feature of HTTP/2 in HAS is firstly proposed by Wei et al [4]. For live streaming, they achieve low latency by reducing the segment duration to one second. To avoid the request explosion problem, they implement the N-push strategy. Specifically, the client requests a certain quality (version) for every N segments. The server then response by pushing N segments consecutively as soon as each one is ready. Besides, the server push feature with N-push strategy has been investigated for low request-related overhead [6] and power efficient mobile streaming [7]. However, by using fixed value of pushed segments for the whole session, the client might not react quickly to network fluctuations. To deal with this problem, Duc et al. [8] define a cost function to adaptively decide the number of pushed segments for each request. This method achieves the tradeoff between request-related overhead and buffer stability. Nevertheless, the bitrate decision in this method is followed the throughput makes the playout bitrate oscillating aggressively, resulting in negative impacts on subjective perception of the user. Moreover, this method just focuses on CBR video.

In this paper, we are the first to have proposed a quality adaptation method in on-demand streaming of VBR video over HTTP/2. We use the cost function in [8] to decide the number of pushed segments. The video version for each request is chosen based on the throughput as well as buffer behavior. The experimental results show that the proposed method

* Corresponding author: Tel.: (+84) 988.980.920
Email: thoa.nguyenthikim@hust.edu.vn

not only can cope with the variations of throughput as well as video bitrate, but also outperforms exiting methods in terms of streaming performance.

The rest of the paper is organized as follows. In Section 2, we present the principles of our method as well as the description of the adaptation algorithm. The experimental results are given in Section 3, and finally the paper is concluded in Section 4.

2. Proposed method

For streaming VBR video over HTTP/2, our goal is to decide the video version and the number of pushed segments for a given request, so as to have a small number of requests, a smooth video quality and a good buffer stability. Some notations along with their definitions used in the paper are provided in Table 1.

Table 1. Symbols used in the paper

Symbol	Description
i	The current request index.
N_i	Number of segments in request i.
j	The current segment index of video, after downloading N_i of request i.
T_i	The actual throughput of segment i.
T_i^e	The estimated throughput for segment i.
$R_{j,n}^a$	The actual bitrate of segment j in version n.
$R_{j,n}^e$	The estimated bitrate for segment j in version n.
R_i^t	the t^{th} segment in request i $(1 \le t \le N_i)$.
V	The number of available video versions
I_i	The index of the version which is chosen for request i (version of higher quality has a higher index value) $(1 \le I_i \le V)$.

Suppose that, after sending request i asking for N_i segments at version I_i, the client has just received all N_i requested segments, each has a segment duration of t seconds. The current buffer level is B_i. Now, the client will decide the version I_{i+1} and the number of pushed segments N_{i+1} for the next request $i+1$.

To select the version for the next request, it is necessary to estimate the throughput based on the throughput history of received segments. Specifically, we adopt the simple method presented in [9] where the throughput of last segment is used as the estimated throughput.

$$T_{i+1}^e = T_i . \quad (1)$$

As the bitrate of VBR video is highly fluctuating, the client only knows the bitrates of the received segments, which may belong to different versions. So, after receiving segment j, we will estimate (i) the segment bitrates of other versions with the same index j and (ii) the segment bitrates at segment $j+1$ of all versions. These are respectively performed by inter-stream bitrate estimation method and intra-stream bitrate estimation method proposed in [10]. Specifically, in the inter-stream bitrate estimation method, the estimated bitrate $R_{j,k}^e$ at segment j of version k can be calculated from the (actual) bitrate $R_{j,n}^a$ of the received segment j with the selected version n as follows:

$$R_{j,k}^e = \varphi \times R_{j,n}^a \times 2^{\frac{QP_n - QP_k}{6}}, \quad (2)$$

where QP_n and QP_k are the quantization parameter (QP) values of the versions, and $\varphi = 1.05$ is an empirical factor used as the compensation for the approximation error of the mothod [10]. In the intra-stream bitrate estimation method, the bitrate of next segment is a function of some previous segment bitrates. It should be noted that, the bitrate estimation of the next segment is required after downloading all segments of each request. Specifically, after downloading N_i segments of a version for request i, the bitrate of the next segment is calculated as the average of the bitrates of N_i previous segments in that version.

For the new bitrate decision, we divide the method into three cases which are switch-up, stable and switch-down cases corresponding to when the client switches up, maintains, and aggressively decreases the version, respectively. Accordingly, we divide the buffer into three ranges with thresholds B_{low} and B_{high} ($B_{low} < B_{high} < B_{\max}$). Here $B_{\max}$ is the buffer size. The details of the proposed method are shown in Algorithm1.

Our method switches up the version by one version only if the current buffer B_i is greater than or equal to B_{high} and the estimated bitrate of next version at the next segment is less than estimated throughput; otherwise, the client will maintain the current version.

The stable case is determined when the current buffer B_i is in the range from B_{low} to B_{high} ($B_{low} \le B_i < B_{high}$). In this case, the current version will be maintained when its bitrate at the next segment is less than estimated throughput; otherwise, the version is switched down by one version.

The switch-down case is described by the condition $B_i < B_{low}$. In this case, the client needs to carefully decide the requested version in order to avoid buffer underflows as well as sudden quality changes when the throughput and/or the video bitrate change drastically. In general, the version should be decreased to the maximum one that is lower than the current version and its bitrate at the next segment is less than estimated throughput. If there is no version that satisfies this condition, the client will choose the lowest version. Specifically, the version decision of the client in this case is expressed as follows:

$$I_{i+1} = \begin{cases} \max\left\{k \middle| k \le I_i, k \in [1;V]\right\}, R^e_{j+1,k} < (1-\alpha)T^e_{i+1} \\ 1, otherwise \end{cases} \quad (3)$$

where α is a safety margin the range [0, 1].

Algorithm1. Adaptation Algorithm

Input B_{low}, B_{high}

Output I_{i+1}, N_{i+1}

```
//Switch-up case
if B_i ≥ B_high then
    if R^e_{j+1,I_i+1} < (1−α)T^e_{i+1}
        I_{i+1} = I_i + 1;
    else
        I_{i+1} = I_i;
    end if
//Stable case
else if B_i ∈ [B_low, B_high)
    if R^e_{j+1,I_i} < (1−α)T^e_{i+1}
        I_{i+1} = I_i;
    else
        I_{i+1} = I_i − 1
//Switch-down case
else B_i < B_low
    if
        for (m = I_i; m ≥ 1; m−−)
            if R^e_{j+1,m} < (1−α)T^e_{i+1}
                I_{i+1} = m;
                break;
            else
                I_{i+1} = 1;
            end if
        end for
    end if
end if
```

For deciding the number of pushed segments, we adopt the method presented in [8]. The number of pushed segments N_{i+1} is chosen to minimize the cost function C which is a weighted sum of request cost C_{rq} and buffer cost C_{bf}.

$$C = \alpha \times C_{rq} + (1-\alpha) C_{bf} \quad (4)$$

with $$C_{rq} = \frac{1}{N_{i+1}} \quad (5)$$

and $$C_{bf} = \frac{N_{i+1} \times t}{B_i - B_{low}}. \quad (6)$$

3. Experimental results

In this section, we will evaluate our proposed method and compare it with the referenced methods [4][8], focusing on the number of client's requests, the average video version and the behaviors of version switching and buffer.

The testbed of our experiments is similar to that of [8], which includes an HTTP/2 web server, an HTTP/2 client and an IP network. The IP network includes a router and a wired connections connecting the server and the client. On the server side, an HTTP/2 enabled server is installed in Ubuntu 14.04 LTS. The client is run in Java environment on a Windows 7 note book with 2.4 GHz and core i5 CPU. The channel bandwidth is simulated using DummyNet [11].

As we focus on on-demand streaming, the buffer size of the client is set to 16s (i.e., 16 segment durations). For comparison, the two reference methods which are the push-N strategy [4] and aggressive method [8] (called AGG method) are implemented. In the push-N strategy, the number of pushed segments is fixed. In the AGG method, the bitrate is decided as the highest bitrate that is lower than estimated throughput.

Experiments are implemented with buffer thresholds $(B_{low}, B_{high} = 5s, 15s)$. Parameters RTT,

a , m are set to 40ms, 0.4, 0.1, respectively. The test video is taken from "Sony Demo" sequence [20], consisting of 500 segments. Each segment duration t is 1 second. The video is encoded in VBR mode at 7 versions (from 1 to 7) corresponding to 7 different QP values which are 48, 42, 38, 34, 28, 22, 10. Note that the higher the value of QP is, the lower the quality (bitrate) become. The version index, QP, and the average bitrate of each version are listed in Table 2. Bitrates of video versions are shown in Fig. 1.

We evaluate the adaptation methods under a time-varying bandwidth obtained from a mobile network [12]. The bandwidth demonstrates a mobile network with strong fluctuations, from 100kbps to 6000kbps (Fig. 2)

Table 2. Version information of the test video.

Index	QP	Average bitrate (kbps)
1	48	29.94
2	42	70.40
3	38	120.85
4	34	207.24
5	28	477.24
6	22	1103.95
7	10	4783.70

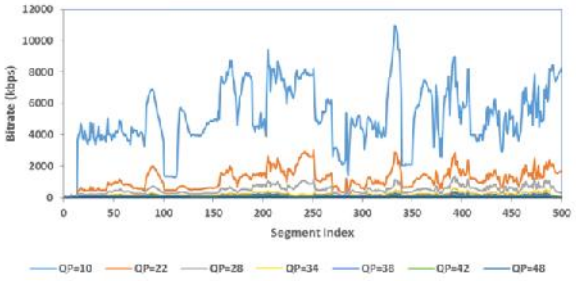


Fig. 1. Bitrates of the video versions

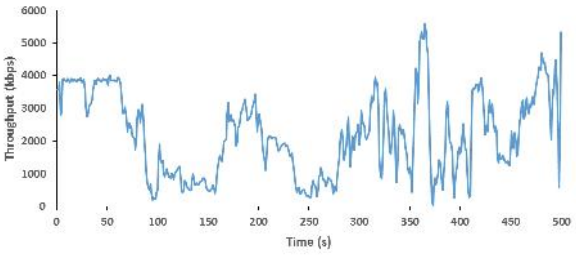
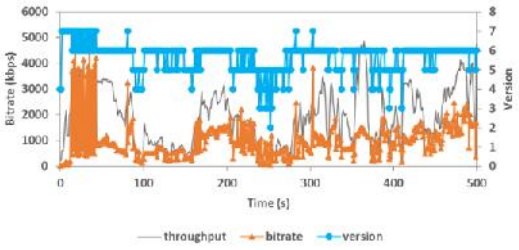
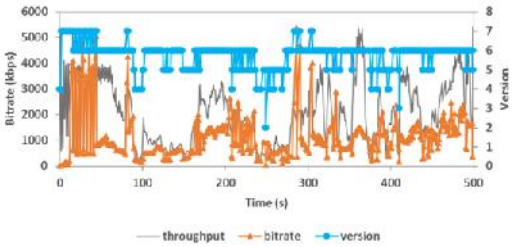


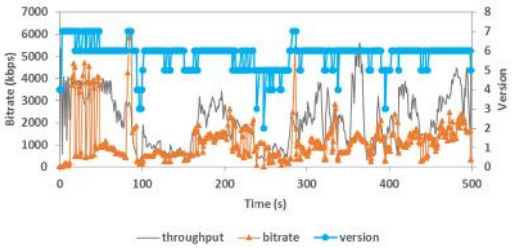
Fig. 2. Bandwidth trace is used in experiments[12]



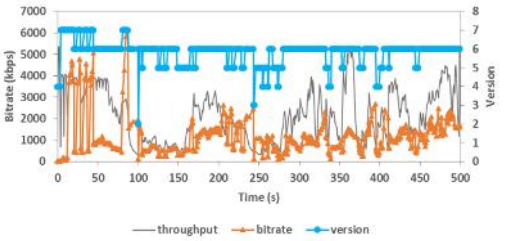
(a) Push 1 strategy



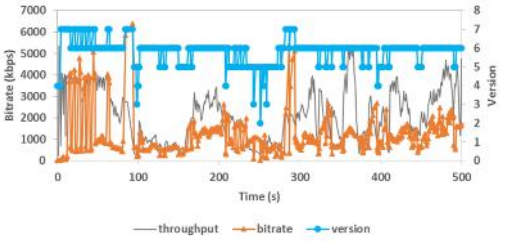
(b) Push 2 strategy



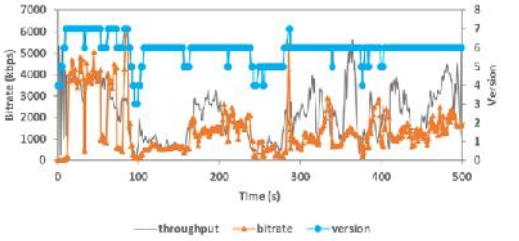
(c) Push 3 strategy



(d) Push 4 strategy



(e) AGG method



(f) Proposed method

Fig. 3. Adaptation results of all methods.

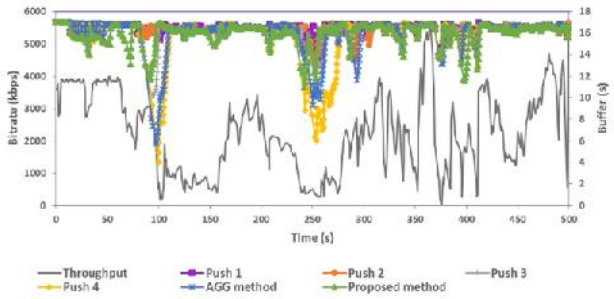


Fig. 4. Resulting buffer level of all methods.

Figure 3 and Fig. 4 show the experimental results of all mentioned methods. Clearly, the push-1 strategy has the most stable buffer. However, its bitrate curve closely follows the throughput curve, leading to the strong fluctuation of video quality with many switches.

As for the push-4 strategy, since it has the large and fixed number of pushed segments per client's request, resulting in sudden drops of quality (e.g., from version 6 to version 2 at 100s). Moreover, it has the most unstable buffer. Specially, at the time 99.5s, the buffer level is less than threshold B_{low}.

It can be seen that the push-2 strategy, the push-3 strategy and the AGG method use the buffer more efficiently leading to the number of switches less than the push-1 strategy. However, they have sudden drops of quality when the throughput sharply decreases (e.g., from version 5 to version 2 at 249s in the push-2 strategy, from version 5 to version 2 at 248s in the push-3 strategy and from version 5 to version 2 at 251s in the AGG method).

Meanwhile, our proposed method provides the smoothest video quality while the buffer stability is acceptable. The version curve of our method has no sudden switches. Moreover, when the throughput decreases, the selected version of our method is usually higher than or equal to those of the other methods.

Some statistics of the adaptation results are shown in Table 3. The statistics show the number of requests, the average bitrate, the quality switches, and the buffer level. It should be noted that we are only interested in the switch quality in down case since it negatively impacts the subjective perception quality of users.

It is obvious that the push-1 strategy has the worst streaming performance. It has the lowest average version (i.e., 5.62) and the most number of switches (i.e., 73). Moreover, in this strategy, network nodes have to handle the highest number of requests (i.e., 500 requests).

Concerning the push-4 strategy, its number of requests is the smallest (i.e., 125). Nevertheless, the buffer stability of this method is the worst.

With regard to the AGG method, it has the better tradeoff between overhead and adaptivity, compared to the push-N strategy. Nonetheless, the maximum version switch degree and the number of switches (having version switch degree greater than 2) are very high (i.e., 3 and 7, respectively).

It can be seen that our proposed method has the best performance compared to the others. Specifically, the proposed method achieves the highest average bitrate and the lowest switch degree while it requires only 173 requests. Furthermore, the number of switches is very low (i.e., 18) and there is no switch having degree exceed 1.

The above results show that the proposed method outperforms the existing methods in terms of the tradeoff between adaptivity and overhead.

Table 3. Statistics of adaptation results

Metrics	Push-N method				Aggressive method	Proposed method
	Push-1	Push-2	Push-3	Push-4		
Number of requests	500	250	167	125	180	173
Average version	5.62	5.68	5.68	5.70	5.74	5.80
Number of switches	73	48	35	26	35	18
Max version switch degree	3	3	3	4	3	1
No. of switches (switch degree>=2)	9	4	5	3	7	0
Min buffer	12.9	12.6	9.9	4.0	5.7	11

4. Conclusion

In this paper, we have proposed an adaptation method for adaptive streaming VBR video over HTTP/2 using the server push approach to reduce significantly number of requests. Besides, we proposed a buffer-based algorithm for deciding the quality level to cope with network and bitrate fluctuations. The experimental results have shown that the proposed method outperforms other exiting methods.

Acknowledgments

This research is funded by the Hanoi University of Science and Technology (HUST) under project number T2016-PC-110.

References

[1] T. C. Thang, Hung T. Le, Anh T. Pham and Y. M. Ro, An Evaluation of Bitrate Adaption Methods for HTTP Live Streaming, IEEE J. Selected Areas in Comm., 32(4), (2014) 693-705.

[2] 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, J. Comm. Networks, 15(6), (2013) 635-644.

[3] S. Wei and V. Swaminathan, Low Latency Live Video Streaming over HTTP 2.0, in Proc. ACM NOSSDAV, (2014) 37-42.

[4] S. Wei and V. Swaminathan, Cost effective video streaming using server push over HTTP 2.0, in Proc. IEEE MMSP'14, (2014) 1-5.

[5] M. Belshe, R. Peon and M. Thomson, Hypertext Transfer Protocol Version 2 (HTTP/2), RFC 7540, May 2015.

[6] 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), (2014) 1-5.

[7] 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), (2015) 1-6.

[8] N. V. Duc, T. H. Le, N. N. Pham, A. T. Pham and T. C. Thang, Adaptation method for Video Streaming over HTTP/2, IEICE communications Express, 1, (2015) 1-6.

[9] L. R. Romero, "A dynamic adaptive HTTP streaming video service for Google Android," M.S. Thesis, Royal Institute of Technology (KTH), Stockholm, Oct. 2011.

[10] T. C. Thang, H. Le, H. Nguyen, A. Pham, J. W. Kang, and Y. M. Ro, Adaptive video streaming over http with dynamic resource estimation, Communications and Networks, Journal of, 15(6), (2013) 635–644.

[11] ServL. Rizzo, Dummynet: A simple approach to the evaluation of network protocols, SIGCOMM Comput. Commun. Rev., 27(1), (1997) 31-41.

[12] C. Muller, S. Lederer, and C. Timmerer, An evaluation of dynamic adaptive streaming over http in vehicular environments, in Proc. of the 4th Workshop on Mobile Video, ser. MoVid '12. New York, NY, USA: ACM, (2012) 37–42.