Cost Effective Video Streaming Using Server Push over HTTP 2.0

Sheng Wei [#1], Viswanathan Swaminathan [#2]

[#] *Adobe Research, Adobe Systems Inc.*
345 Park Ave, San Jose, CA 95110, USA
[1] swei@adobe.com
[2] vishy@adobe.com

***Abstract*—The Hypertext Transfer Protocol (HTTP) has been widely adopted and deployed as the key protocol for video streaming over the Internet. One of the consequences of leveraging traditional HTTP for video streaming is the significantly increased request overhead due to the segmentation of the video content into HTTP resources. The overhead becomes even more significant when non-multiplexed video and audio segments are deployed. In this paper, we investigate and address the request overhead problem by employing the server push technology in the new HTTP 2.0 protocol. In particular, we develop a set of push strategies that actively deliver video and audio content from the HTTP server without requiring a request for each individual segment. We evaluate our approach in a Dynamic Adaptive Streaming over HTTP (DASH) streaming system. We show that the request overhead can be significantly reduced by using our push strategies. Also, we validate that the server push based approach is compatible with the existing HTTP streaming features, such as adaptive bitrate switching.**

I. Introduction

The Hypertext Transfer Protocol (HTTP) streaming has been widely adopted and deployed for delivering video content over the Internet in the recent years. The HTTP streaming technologies leverage the existing web infrastructures and resources to serve video content in a scalable and easy-to-deploy manner [1][2]. However, since the HTTP protocol was not originally invented, designed, or optimized for video streaming, it requires adaptations at the application level to enable a cost effective streaming solution.

One of the major costs in HTTP based video streaming is due to the segmentation mechanism, where the video content is packaged into multiple segments that can be deployed as the basic unit for HTTP requests and responses. In order to obtain the video content, the client must explicitly request each individual segment, in the form of a HTTP request, from the server. Different from traditional web applications (e.g., web page browsing), the request overhead in HTTP video streaming is proportional to the time of user engagement, i.e., the duration of the video. To make things worse, in a typical use of HTTP streaming, the video and audio content is often not multiplexed to lower the storage overhead [3], which applies to all cases where multiple audio streams are supported for the same video stream. Consequently, the number of segments and hence the number of requests is doubled compared to multiplexed audio/video streams.

The large number of HTTP requests to download the video segments introduces significant costs. For example, from the perspective of the client, the overhead in establishing the connections and sending the requests frequently poses a challenge on the resource constraints. The challenge becomes even more significant with the popularity of video streaming onto power constrained mobile devices, as discussed in [4] and [5]. In [4], the authors explicitly showed that the frequent requests of small segments consume significantly more battery on a mobile phone than downloading multiple segments in batch. To further validate this observation, we also conducted experiments on a mobile phone and a laptop computer comparing the battery performance under the scenarios of frequent download and bundled download. The details of our experimental setup and results can be found in [6], which validated the findings of [4]. In summary, both the literature and our experiments show that the frequent requests of video segments cause battery inefficiency, which is considered as a major source of cost for mobile devices.

While we are seeking for a cost effective HTTP streaming solution, the HTTP 2.0 protocol has been actively developed and planned for standardization [7]. One of the new features proposed in HTTP 2.0 is *server push*, where the server can maintain a persistent connection with the client and actively push multiple resources without requiring a request for each individual resource. The server push feature in HTTP 2.0 was designed for web page browsing, where embedded resources in a HTML page can be actively pushed to the client to enable a faster web experience. However, we observe that our video streaming scenario can possibly benefit from the server push feature, since the resources that may be requested by the client are deterministic, which are pre-defined in the manifest file. If the server actively pushes the pre-defined video and audio content to the client, the request overhead can be significantly reduced.

We investigate and develop a set of server push strategies for cost effective video streaming over HTTP 2.0. Our key push strategy, namely audio push, is for the server to automatically push the audio segment upon receiving the request for a video

2014 IEEE 16th International Workshop on Multimedia Signal Processing (MMSP), Sep. 22-24, 2014, Jakarta, Indonesia.
978-1-4799-5896-2/14/$31.00

segment, given that the client would always request the audio segment that corresponds to the video segment. Then, we further extend the audio push strategy by allowing the server to push additional audio and video segments. While enabling and demonstrating the request reduction achieved by server push, we pay close attention to the additional overhead introduced by server push, such as delays in adaptive bitrate switching. To summarize, our key contributions in this paper include the following:

- we for the first time employ the server push technology for cost effective non-multiplexed video streaming;
- we develop a set of audio and video push strategies for streaming over HTTP 2.0, which significantly reduces the request overhead; and
- we evaluate the benefits and overhead of the server push strategies.

II. Related Work

HTTP has been widely adopted as the preferred protocol for entertainment video streaming over the Internet because of its simplicity and scalability [1][2]. However, the power consumption at the client while watching the video is a significant concern. Particularly, the large number of video and audio segments periodically requested by the client for each video session results in significant power usage. These segments also result in costs at the content delivery network (CDN) caches. Tian et al. [4] studied the consequences of HTTP adaptive streaming in battery consumption on mobile devices. They developed a bundled chunk download schedule to reduce the power cost. Hoque et al. [5] provides a comprehensive survey on energy efficient multimedia streaming solutions across all the protocol layers. They concluded that the energy consumption characteristics in HTTP adaptive streaming scenario still need further explorations.

Recently, the advances in HTTP 2.0 [7][8][9] have drawn a great deal of attention. There have been several works that focus on evaluating the improved web access performance, such as latency, over HTTP 2.0 [10][11][12]. In 2013, Mueller et al. [13] evaluated the performance of DASH video streaming over HTTP 2.0. In early 2014, our group developed a low latency live streaming solution leveraging the server push feature in HTTP 2.0 [14], which suggests the use of small segments together with server push, to reduce live latency without increasing the request overhead. Following our prior work, this paper is the first attempt leveraging HTTP 2.0 to reduce the request overhead in on-demand video streaming.

III. Server Push based HTTP Video Streaming

A. Overall Flow and Architecture

Figure 1 shows the overall flow of our server push based HTTP streaming scheme. With a HTTP 2.0 connection, the server can actively push certain video or audio segments, which have not been requested by the client, following a server push strategy. With this mechanism, although there are more segments sent by the server for each request, the stateless nature of each request is preserved. The client maintains a local cache that can receive and store the pushed content temporarily. During the video session, once the client requests a video or audio segment, it first searches the local cache to see if the particular segment has been pushed by the server. If the search succeeds, the client retrieves the segment from the cache without sending the HTTP request over to the server; otherwise, the client sends a regular HTTP request to the server, and the server responds with the requested segment in a regular HTTP response.

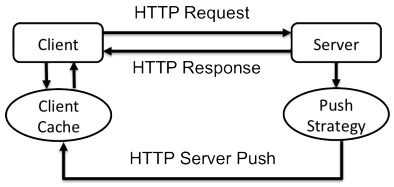

Fig. 1. Overall flow of server push based HTTP video streaming.

B. Push Strategy for Non-multiplexed Video Streaming

The most important component in the server push based streaming scheme discussed in Figure 1 is the server push strategy, as it specifies when the server should push content to the client, as well as which video or audio segments to push. The design of the push strategy is application-dependent. For example, the referrer based strategy for web page browsing [16], which pushes the embedded resources in a requested HTML file, does not apply to the video streaming applications, as there are no referrer relationships between various independent video or audio segments. In the design of our push strategy for video streaming, we take into account the following principles. First, we aim to minimize the regular HTTP requests that are actually sent by the client to the server. Second, we must ensure that the applied push strategy, and consequently the resulting video streaming scheme, is still compatible with the key HTTP video streaming features, such as adaptive bitrate streaming. Third, we must minimize the additional overhead introduced by the server push strategy.

Following the above design principles, we develop our server push strategies as shown in Figure 2. Figure 2(a) shows the no-push case over HTTP 1.1, which we consider as a baseline for comparison with our approach. Figure 2(b) is a push strategy for pushing audio segments only, in which the server pushes the corresponding audio segment upon receiving the request of a video segment. Figure 2(c) shows the K-push strategy for pushing both video and audio segments, in which the server pushes $K-1$ consecutive pairs of audio/video segments ($K > 1$), along with the associated audio segment, upon receiving the request for a video segment.

C. Overhead in Server Push over HTTP 2.0

While analyzing the push strategies, we observe two possible sources of additional overhead that may be introduced in the video streaming. Both of them may occur in the scenario of bitrate switching, when the video player decides to switch to a different bitrate of audio/video segments while monitoring

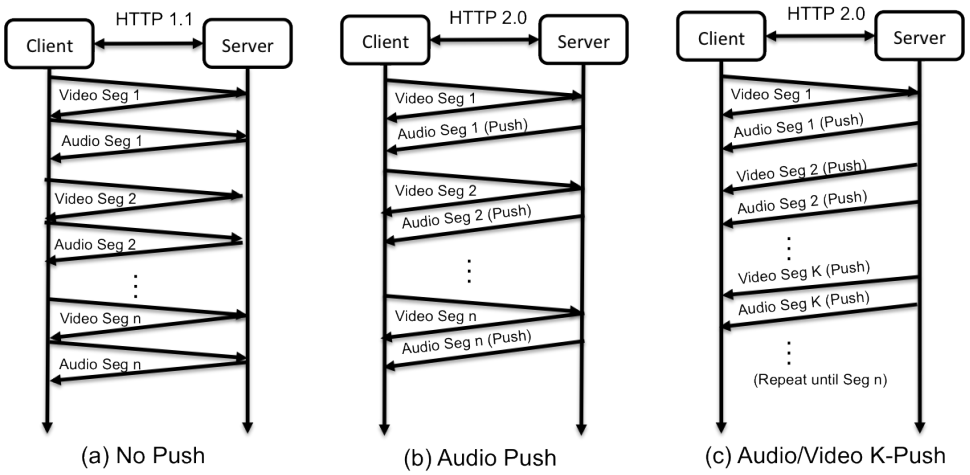

Fig. 2. Push strategies for HTTP video streaming.

the bandwidth condition. The first possible overhead is the bitrate switching delay, which is defined as the delay between the point when the client decides to switch to a new bitrate and the point when it actually receives the segment of the desired bitrate. It is important to note that the switching delay is strongly dependent on the network condition, and it may increase significantly due to the existence of the pushed stream that consumes the bandwidth.

The K-push strategy may introduce another source of overhead, namely bandwidth overhead, which may be caused by the already pushed content of the old bitrate. Since the client is switching to a new bitrate, this set of segments would never be consumed by the client, and thus they are considered as the unnecessary consumption of the network bandwidth. Furthermore, following the HTTP 2.0 draft specification [7], the client can cancel the pushed stream at any time by issuing the "CANCEL" stream error to minimize the bandwidth overhead. However, due to the limitation in the current implementation of the HTTP 2.0 stack, we leave the issue of the additional bandwidth overhead as future work.

IV. Implementation and Experimental Results

A. System Design Experimental Setup

Figure 3 shows the architecture of our experimental prototype, which is a DASH video streaming system over HTTP 2.0. The prototype has three major components: (1) The client is the DASH.js player [17] running on the Chrome browser [18], which supports Google SPDY [8], the draft version of HTTP 2.0; (2) the cache server, which we implemented using a modified version of the Jetty web server [16] deployed with SPDY support and our server push strategies; and (3) the origin server, which is a regular web server that supports HTTP 1.1. The DASH sequence we use in the experiment is the multi-representation main profile sequence provided by Telecom ParisTech [15]. We deploy a subset of the sequence on the origin server, which contains 30 10-second video segments in 4 different representations (viz, low - 51 Kbps, mid - 195 Kbps, hd - 515 Kbps, and high - 771 Kbps), as well as 32 9.52-second audio segments in 2 representations (viz, low - 19 Kbps and high - 66 Kbps). We control the network bandwidth and delay using the network link conditioner in Xcode [19].

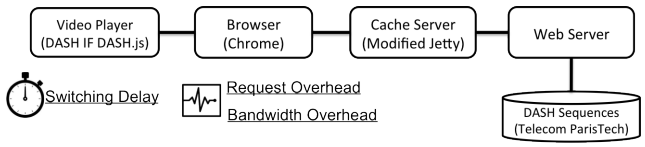

Fig. 3. Experimental system design.

Based on the experimental platform, we conduct two sets of measurements. First, in both the single and multi-bitrate streaming scenarios, we measure the request overhead at the video player, in the form of number of requests issued by the client during the video playback. Second, during bitrate switching, we measure the bandwidth overhead (in the form of number of unclaimed HTTP pushes) as well as the bitrate switching delays. In our experiments, we collect the overhead results once all the required resources are already cached in the cache server, which excludes the bandwidth and delay overhead caused by the communication between the origin and cache servers.

B. Request Overhead

We first run our experiments under a single-bitrate video streaming scenario and measure the request overhead savings obtained from the audio push and audio/video K-push schemes. Figure 4 [1] shows the experimental results comparing different push strategies. [2] We observe that audio push saves

[1] Although the presented figure may be derived analytically, we conducted experiments to validate that our experimental setup is real and reflective of theoretical calculation.

[2] In this experiment, we specifically focus on the request overhead savings obtained from the server push feature in HTTP 2.0 and do not cover possible savings that may be obtained by other features, such as the persistent connection and stream multiplexing.

around 50%, and audio/video K-push saves over 70%, in the request overhead as compared to the no-push case. Also, the savings grow with the parameter K. In the extreme case of K=30, which means that the server pushes all the possible segments upon receiving the first request, the number of requests drops to less than 10 for the entire video. [3].

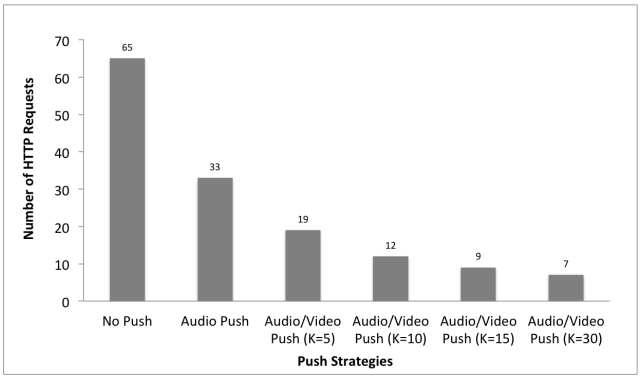

Fig. 4. Request overhead in single-bitrate video streaming.

C. Overhead in Multi-bitrate Switching

We evaluate the overhead in our push strategies during bitrate switching using two metrics, namely switching delay and unclaimed pushes. Table I shows the switching delay when the video streaming switches from a high bitrate to a low bitrate under certain network bandwidth and delay (i.e., 1 Mbps downlink bandwidth and 5 ms network delay). The video segment duration is 10 seconds. We observe that the audio push strategy introduces only slightly higher switching delay compared to the no-push case. In the audio/video K-push case, the switching delay increases significantly due to the fact that the pushed stream exists and consumes the available bandwidth at the time of switching and, therefore, both the requests and responses for the desired new segments are delayed. This constitutes the major source of overhead for using audio/video K-push. The results show that the switching delays are around the boundary of 2 segment durations and are independent of the K values. This is because the pushed stream and the request/response of the new segment share the same HTTP 2.0 persistent connection, and the server can deliver the new segment after the completion of the currently pushed segment and before the next push.

Furthermore, Figure 5 shows the request overhead and bandwidth overhead during a video session with varying bandwidth configurations (i.e., bandwidth reduces from 2 Mbps to 1 Mbps during the video session [4]). We observe that the parameter K plays an important role in the gains and overhead. The results indicate a possible best choice for a low overhead push strategy is audio push, which introduces only slightly higher overhead than no-push but with significant gains in the number of requests.

TABLE I
BITRATE SWITCHING DELAYS USING NO-PUSH, AUDIO PUSH, AND AUDIO/VIDEO K-PUSH (WITH BANDWIDTH CONFIGURED AT 1 MBPS, VIDEO SWITCHING FROM "HIGH" TO "HD", AND VIDEO SEGMENT DURATION OF 10 SECONDS).

Push Strategy	Switching Delay (sec)
No Push	6.474
Audio Push	6.520
Audio/Video Push (K=5)	19.898
Audio/Video Push (K=10)	19.919
Audio/Video Push (K=15)	19.877
Audio/Video Push (K=30)	21.041

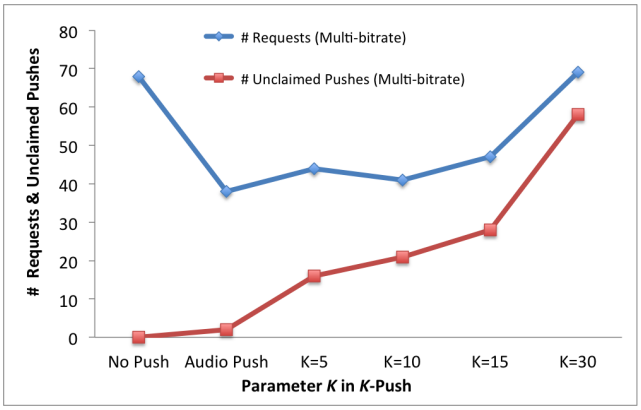

Fig. 5. Request and bandwidth overhead in multi-bitrate video streaming (with bandwidth varying from 2 Mbps to 1 Mbps).

V. CONCLUSIONS

We have developed a server push based approach over HTTP 2.0 to reduce the request overhead in HTTP video streaming, especially in the case where video segments and audio segments are non-multiplexed. We developed audio push and audio/video K-push strategies that actively deliver video segments from the server to the client without requiring an explicit HTTP request for each segment. We implemented and evaluated the proposed approach in a DASH based video streaming system. Our experimental results indicated at least around 50% savings in the request overhead depending on the push strategy and the parameter settings. Also, we show that the additional overhead caused by the server push based approach, namely multi-bitrate switching delay and bandwidth overhead, can be well controlled by tuning the parameter K in the K-Push strategy. As future work, we will continue to evaluate the throughput degradation in the average and worst cases, optimal K selection in the K-push strategy, and experiment with the scalability issues at the origin and cache servers.

[3]Here the issued requests include the initial requests of the manifest files and the first video segment request, which are not registered as pushed resources in our experimental setup.

[4]In our experiments, we observed that the video player did not select a bitrate that uses 100% of the bandwidth and, therefore, it would switch to a lower bitrate even if 1Mbps appears to be an enough bandwidth for the current bitrate. We consider this as a configuration detail of the player, which does not impact our experimental results.

REFERENCES

[1] I. Sodagar, The MPEG-DASH standard for multimedia streaming over the Internet, IEEE MultiMedia, Vol.18, No. 4, pp. 62-67, 2011.

[2] V. Swaminathan, Are we in the middle of a video streaming revolution? ACM Transactions on Multimedia Computing, Communications and Applications, Vol. 9, No. 1s, Article 40, 2013.

[3] S. Mitra, V. Swaminathan: An optimal client buffer model for multiplexing HTTP streams, International Workshop Multimedia Signal Processing (MMSP), pp. 283-288, 2012.

[4] G. Tian, Y. Liu, On adaptive HTTP streaming to mobile devices, Packet Video Workshop, pp. 1-8, 2014.

[5] M. Hoque, M. Siekkinen, J. Nurminen, Energy efficient multimedia streaming to mobile devices-A survey, IEEE Communications Surveys & Tutorials, Vol. 16, No. 1, pp. 579-597, 2014.

[6] S. Wei, V. Swaminathan, Battery tests on mobile devices, Technical Report, May 2014, available online: http://videosystemshub.com/doc/TechReport/BatteryTests.pdf

[7] M. Belshe et al., Hypertext Transfer Protocol version 2, draft-ietf-httpbis-http2-13, HTTPbis Working Group, June 2014, available online: http://tools.ietf.org/html/draft-ietf-httpbis-http2-13

[8] SPDY: An experimental protocol for a faster web, available online: http://www.chromium.org/spdy/spdy-whitepaper

[9] Hypertext Transfer Protocol Bis (httpbis) - Charter, IETF, 2012, available online: https://datatracker.ietf.org/wg/httpbis/charter/

[10] A. Cardaci, L. Caviglione, A. Gotta, N. Tonellotto, Performance evaluation of SPDY over high latency satellite channels, International Conference on Personal Satellite Services (PSATS), pp. 123-134, 2013.

[11] G. Mineki, S. U emura, T. Hasegawa, SPDY accelerator for improving web access speed, International Conference on Advanced Communication Technology (ICACT), pp. 540-544, 2013.

[12] J. Padhye, F. Nielson, A comparison of SPDY and HTTP performance, Microsoft Research Technical Report, MSR-TR-2012-102, 2012, available online: http://research.microsoft.com/apps/pubs/?id=170059

[13] C. Mueller, S. Lederer, C. Timmerer, H. Hellwagner, Dynamic adaptive streaming over HTTP/2.0, International Conference on Multimedia and Expo (ICME), pp. 1-6, 2013.

[14] S. Wei, V. Swaminathan, Low latency live video streaming over HTTP 2.0, ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), pp. 37-42, 2014.

[15] GPAC DASH sequences, available online: http://gpac.wp.mines-telecom.fr/2012/02/23/dash-sequences/

[16] Jetty HTTP server, available online: http://www.eclipse.org/jetty/

[17] dash.js, DASH Industry Forum, available online: https://github.com/Dash-Industry-Forum/dash.js

[18] The Chromium projects, available online: http://www.chromium.org

[19] Designing for real-world networks, iOS Developer Library, TP40010220-CH13-SW1, 2014.