Live Streaming of 4K Ultra-High Definition Video over the Internet

Stefano Petrangeli[*], Jeroen van der Hooft, Tim Wauters
Ghent University - iMinds

Rafael Huysegems, Patrice Rondao Alface, Tom Bostoen
Nokia Bell Labs

Filip De Turck
Ghent University - iMinds

ABSTRACT

HTTP Adaptive Streaming (HAS) is the de facto standard for video streaming services over the Internet. In HAS, each video is temporally segmented and stored in different qualities. The client selects the quality level for every video segment based on network conditions, allowing a smooth playback with the best possible Quality of Experience (QoE). Although results are promising, current solutions suffer from two problems. First, a low quality and large end-to-end latency are often observed in live streaming scenarios. Second, freezes in the video playout may occur in case of sudden drops of the available bandwidth. We reduced these issues using two complementary approaches. First, we reduced the live latency using the new HTTP/2 server push in combination with super-short segments. Second, we designed an OpenFlow-based network controller that prioritizes the delivery of particular segments to avoid freezes at the clients. The proof-of-concept shows the results obtained when two clients stream a video under varying network conditions. By monitoring the clients' behavior, it is possible to understand the gains brought by the proposed approaches. Particularly, we demonstrate how our solutions consistently reduce the live latency in high round-trip time networks and video freezes caused by network congestion. These results represent a major improvement for the QoE of the final users.

CCS Concepts

•General and reference → Design; •Information systems → Multimedia streaming; •Networks → Network management; Wide area networks;

Keywords

HTTP Adaptive Streaming, Live Latency, HTTP/2, Server Push, Video Freezes, OpenFlow, Prioritization, Quality of Experience

[*]Corresponding author: Technologiepark 15, B-9052 Ghent, Belgium. E-mail: stefano.petrangeli@intec.ugent.be

XXX XXX
© 2016 ACM. ISBN XXX…$15.00
DOI: XXX

1. INTRODUCTION

Nowadays, video streaming applications account for the largest portion of the Internet traffic. Particularly, HTTP Adaptive Streaming (HAS) protocols have become very popular and are considered to be the de facto standard for video streaming services over the Internet. Microsoft's Smooth Streaming, Apple's HTTP Live Streaming, Adobe's HTTP Dynamic Streaming and MPEG's Dynamic Adaptive Streaming over HTTP (DASH) are examples of deployed HAS technologies. In an HAS architecture, video content is temporally segmented at different quality levels and stored on a server. Each client requests these segments at the most appropriate quality level, based on the locally perceived bandwidth and the video player buffer level. In this way, video playback dynamically changes according to the available resources. Such dynamic adaptation results in a smooth video streaming experience.

Nevertheless, several inefficiencies still have to be solved in order to further improve users' Quality of Experience (QoE). For instance, the camera-to-display delay, which is the delay between capturing an event and its playout on the client's display, is very important when streaming live events. In current HAS deployments however, this delay is in the order of tens of seconds. This is because a large buffer at the client is required to prevent playout freezes and the server only sends a new video segment once a request is issued by the client. Moreover, as reported by Akshabi et al., current rate adaptation heuristics perform quality selection sub-optimally, especially when a sudden bandwidth drop occurs [1]. This leads to unnecessary quality switches and video playout interruptions, which negatively affect the overall QoE of the users. Similar conclusions are drawn in the 2015 Conviva report on HAS [3]. The report reveals that almost 29% of the analyzed HAS sessions exhibit at least one video freeze. This problem is mainly due to the unmanaged nature of current HAS technologies, as the clients are only aware of the local perceived bandwidth conditions and cannot be assisted in improving the delivered QoE. This issue is aggravated in case of live streaming, where the client buffer has to be kept as small as possible in order to reduce the live latency.

We solved the aforementioned problems using two complementary approaches. First, we reduced the live latency using the server push feature of the HTTP/2 protocol together with super-short segments with a duration of 100 to 500 ms, compared to traditional values of at least 1 second. Super-short segments allow reducing the client buffer and, consequently, the live latency. Unfortunately, the number of HTTP GET requests increases in this case, resulting in a large network and server overhead and a lower bandwidth utilization in high round-trip time (RTT) networks, such as wireless networks. By actively pushing segments from the server to the clients using the HTTP/2 protocol, it is possible to eliminate these drawbacks. Second, we designed an OpenFlow-based framework to help the clients avoiding video freezes caused by network congestion. The main element of this framework is a controller, which prioritizes the delivery of particular HAS segments in order to avoid video freezes. This decision is based on the HAS clients' status and on measurement data collected from the network nodes.

Within the V-FORCE[1] project, a Proof-of-Concept (PoC) has been developed to demonstrate the gains brought by the proposed approaches. The demo is composed by two video clients, streaming a video simultaneously. By varying the available bandwidth and RTT, we can demonstrate that (i) our HTTP/2 solution significantly reduces the live latency and startup delay in high-RTT networks and (ii) network-based prioritization can effectively prevent the occurrence of freezes, without decreasing the video quality.

The remainder of this paper is structured as follows. Section 2 reports related work on HAS optimization. Next, Section 3 details the proposed HTTP/2 push-based solution and the OpenFlow-based framework. In Section 4, the demo setup is presented, while Section 5 concludes the paper.

2. RELATED WORK

Wei et al. are the first to explore how the new HTTP/2 push feature can be used to improve HAS [10]. By reducing the segment duration from five seconds to one second, they reduce the camera-to-display delay to about ten seconds. They avoid an increased number of GET requests by pushing k segments after each request, using HTTP/2 server push. This approach has the disadvantage that when a client switches to another quality level, the push stream for the old quality level is in competition with the segments downloaded at the new quality level. Cherif et al. propose DASH fast start, in which HTTP/2's server push is used to reduce the startup delay in a DASH streaming session [2]. The adaptation logic is also moved from the client to the server. In our approach instead, the classical HAS principle is not altered.

The use of an OpenFlow controller to optimize the behavior of the HAS clients has been studied by Egilmez et al. [4]. They propose to dynamically re-route HAS traffic to avoid congested links. As traffic re-routing is only possible in the core Internet Service Provider network while congested links mainly arise in the edge network, this approach is not able to fully optimize the behavior of the HAS clients. Several other works apply traffic-shaping techniques to limit the bandwidth assigned to each client and to drive them to request a target bit rate [5, 6]. This entails that the network de-facto decides which quality level the clients can download. Moreover, this approach is not designed to foresee the occurrence of video freezes and avoid them. In our solution instead, the quality level decision is completely left to the clients. The OpenFlow controller supports the delivery of particular segments to avoid a freeze but does not have any active role in the quality decision process of the clients.

[1]https://www.iminds.be/en/projects/2014/09/23/v-force

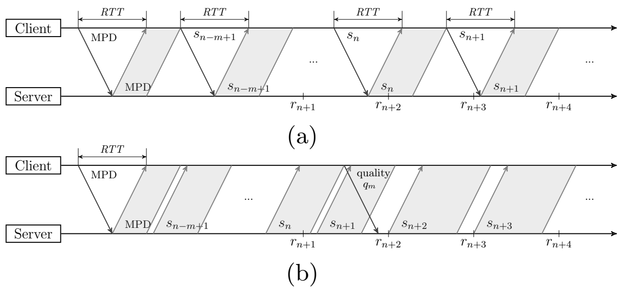

Figure 1: An example live video scenario for HTTP/1.1 (a) and HTTP/2 (b), where the client requests m available segments to ramp up the buffer. Note that r_i denotes the release of segment i at server side, s_i its request/download by the client and q_j indicates that the server should change the quality of pushed segments to j.

3. IMPROVED DELIVERY OF LIVE VIDEO

In this Section, we highlight the main elements and components of our HTTP/2 based solution and OpenFlow-based prioritization framework. We refer to our previous works by Huysegems et al. and Petrangeli et al. for a more extensive view on the matter [7, 9].

3.1 Reducing the live latency with HTTP/2

In HAS, a video streaming session starts with the client sending a request for the video's media presentation description (MPD). Based on the contents of the MPD, the client requests video segments one by one, typically ramping up the buffer by downloading segments at the lowest quality. The main drawback of this approach is that multiple RTT cycles are lost to download the MPD and the segments, which has a significant impact on the startup time in high-RTT networks. An illustration of this behavior is shown in Figure 1a, where the first phase of an HTTP/1.1 live streaming session is shown.

In the HTTP/2 full-push approach, the server pushes m segments to the client as soon as the MPD request is received, where m corresponds to the number of segments that fit into a preferred buffer size defined by the client. Since state-of-the-art rate adaptation heuristics ramp up the buffer by downloading segments at the lowest quality, these segments are pushed at the lowest quality as well. Note that the client cannot select another quality anyway, since the MPD defining all quality levels has not yet been received. As illustrated in Figure 1b, at least one RTT cycle is gained in the reception of the first video segment, and multiple RTT cycles are gained during the buffer ramp up phase. Once the manifest and the first m segments are sent, the server periodically pushes a new segment to the client at the specified quality level, whenever a new segment is available. At any time, the client can send a request to change the bit rate of pushed segments, if required. The proposed approach has a number of advantages. Since the first m segments are pushed back-to-back when the manifest is requested, the client's startup delay can be significantly reduced in high-RTT networks. Since no RTT cycles are lost to request the video segments, super-short segments can be used, which further reduces the startup delay. Using a smaller buffer, the approach can effectively reduce the overall camera-to-display delay as well.

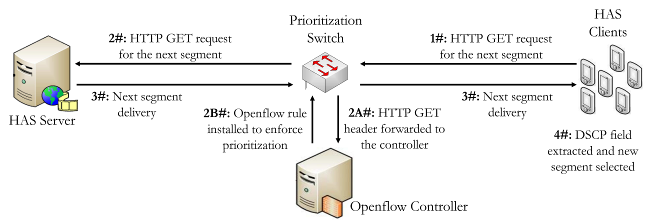
Prioritization Switch
HAS Clients
2#: HTTP GET request for the next segment
1#: HTTP GET request for the next segment
HAS Server
3#: Next segment delivery
2B#: Openflow rule installed to enforce prioritization
2A#: HTTP GET header forwarded to the controller
3#: Next segment delivery
4#: DSCP field extracted and new segment selected
Openflow Controller

Figure 2: The OpenFlow controller intercepts clients' requests and decides whether the requested segment should be prioritized or not.

3.2 Reducing video freezes with OpenFlow

In order to help clients avoiding freezes in case of bandwidth drops, we designed an OpenFlow controller that prioritizes the delivery of video segments. Prioritization is enforced in the network by using an OpenFlow-enabled switch, the so-called prioritization switch, which is equipped with a best-effort and a priority queue. An illustrative sequence diagram of the proposed framework is shown in Figure 2.

The OpenFlow controller intervenes each time a client requests a new segment from the HAS server and decides whether the analysed segment should be prioritized or not. To perform this decision, the controller obtains relevant measurements from both the prioritization switch and the HAS client requesting the new segment. Network measurements are obtained through the OpenFlow protocol, which provides well-defined APIs to collect data from the OpenFlow switches. More specifically, the controller periodically polls the prioritization switch to compute the average throughput of the best-effort and prioritized queue (not shown in Figure 1). Given the overhead and complexity of implementing a direct communication channel between the clients and the controller, client related measurements are transmitted by introducing an additional field in the header of the HTTP GET message sent by the client when requesting a new segment. The information signalled by the clients is the current buffer filling level and the size and duration of the requested segment. The prioritization switch is configured to forward the HTTP GET header to the controller via an OpenFlow rule. The decision on which segment to prioritize is carried out by computing an estimate of the segment download time in the best-effort and prioritized queue. If a best-effort delivery does not guarantee a timely download of the segment, i.e., if the download time is larger than the client buffer filling level, the segment is prioritized. Next, the controller installs a new OpenFlow rule on the prioritization switch to guarantee the proper delivery of the analysed segment.

In our solution, the clients are aware of the prioritization status of the downloaded segments. The prioritization switch is configured to mark prioritized packets with a specific DSCP field. This field is extracted by the clients during the download of a segment to discover whether the segment was prioritized or not. In case of prioritization, the segments is ignored in the calculation of the estimated bandwidth because the bandwidth perceived in case of prioritization does not match the real network conditions. In addition, the client directly requests the next segment at the lowest quality in order to minimize the risk of freezes, which is high as the prioritization indicates. It is worth noting that this mechanism is independent from the actual adaptation heuristic implemented by the client.

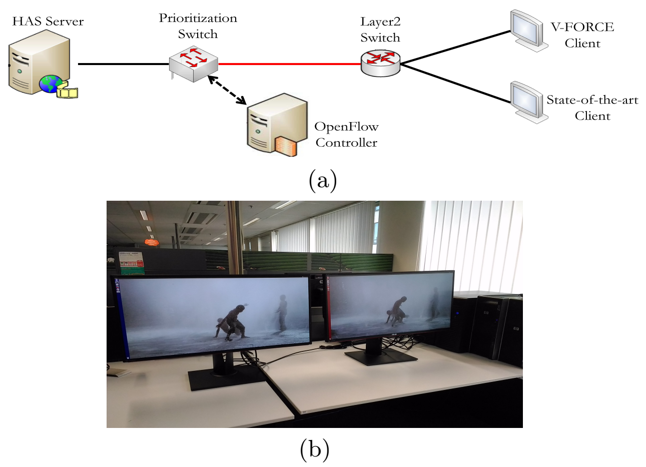

Figure 3: The PoC is composed by a video server, an OpenFlow switch connected to an OpenFlow controller, a layer 2 switch and two video clients.

4. PROOF-OF-CONCEPT SETUP

The network setup of the V-FORCE PoC is composed by an HAS server, two clients, an OpenFlow switch controlled by an OpenFlow controller and a layer 2 switch (Figure 3).

The HAS server implementation is based on the Jetty web server, which was recently extended to provide support for HTTP/2. Jetty's HTTP/2 component implements a push-based strategy, which defines all resources that need to be pushed along with the requested resource. Such a strategy is ideal for web-based content but not for a live-stream scenario, since not all segments are available when a request is issued. Therefore, we defined a new request handler that processes GET requests issued by the client. This handler allows a client to issue a live-stream request, passing along parameters such as the maximum buffer size and quality level. When this request corresponds to a new session, the server starts a push thread that pushes the m last released video segments at the lowest quality immediately. In order to simulate a live stream scenario, a release thread makes new segments available every segment duration. As soon as a new segment is available, the push thread is notified and a segment is pushed to the corresponding client. When the client wants to change the quality level at which the segments are pushed, a new GET request is issued and the quality level is updated at server-side accordingly.

The HAS client is implemented on top of the libdash library, the official reference software of the ISO/IEC MPEG-DASH standard. To make use of the server push provided by HTTP/2, a number of changes is made. First, an HTTP/2-based connection is added to enable the reception of pushed segments. The nghttp2 library is used to set up an HTTP/2 connection over SSL. Note that the reference software uses curl to issue all GET requests, yet this library does not yet support HTTP/2 push. Second, the rate adaptation heuristic is modified to recalculate the quality level every time a segment is received and the corresponding push stream is closed. While in HTTP/1.1 a GET request is required for every segment, no request is sent in the HTTP/2-based scheme if no quality change is required. Third, the perceived bandwidth is estimated based on the elapsed time between the reception of the push promise and the time the segment is available. The Libpcap library is used to extract the DSCP field from the received packets and thus enable prioritization

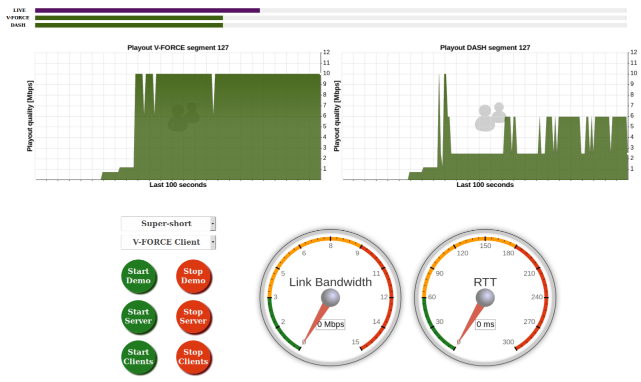

Figure 4: The HTML5 dashboard allows to fully control the demo setup and monitor the performance of the video clients.

awareness. The rate adaptation heuristic embedded into the HAS clients is the FINEAS algorithm [8].

The OpenFlow controller is implemented using POX, an extendible Python-based controller. Open vSwitch is used to realize the prioritization switch, which is equipped with a best-effort and a priority queue.

The considered video is the 60 fps Netflix El Fuente sequence, encoded at variable bit rate using H.265/HEVC in two different versions with 2000 ms and 500 ms segment durations. Seven quality levels are provided, with a nominal bit rate ranging from 0.3 Mb/s to 20 Mb/s, and spatial resolution from 720p to 4K. In order to guarantee the same visual quality for the 2000 ms and 500 ms cases, the segments were encoded with very close global Peak Signal to Noise Ratio (PSNR) values. With PSNR differences smaller than 0.2 dB, the encoding rates for the 500 ms case are on average 15% higher than in the 2000 ms case. This overhead is due to the encoding cost of four times more frequent instantaneous decoder refresh (IDR) frames.

An HTML5 dashboard allows to control the network setup and monitor the behavior of the HAS clients (Figure 4). By using the dashboard, it is possible to dynamically change the bandwidth and the RTT on the link connecting the two switches (highlighted link in Figure 3a). As for the monitoring, three metrics can be visualized: (i) the quality downloaded by the client, (ii) the index of the downloaded segment and (iii) the index of the latest released segment by the server. The latter two metrics indicate the playout delay between the client and the live signal, i.e., the live latency. In a first scenario, we demonstrate that our V-FORCE client experiences a lower live latency and a higher quality compared to the state-of-the-art client when the network RTT is high, thanks to the HTTP/2 server push feature and super-short segments. In a second scenario, we introduce drops in the available bandwidth and show how the prioritization introduced by the OpenFlow controller can reduce the occurrence of video freezes for the V-FORCE client.

5. CONCLUSIONS

In this paper, we presented the proof-of-concept developed within the V-FORCE project, which allows us to demonstrate two main points. First, the server push feature of the HTTP/2 protocol can effectively decrease the startup delay and live latency in high-RTT networks, when combined with super-short segments. Second, network-based prioritization introduced by an OpenFlow controller can reduce video freezes caused by network congestion. Monitoring information allow to compare the performance of the V-FORCE optimized client and a state-of-the-art client. Practically, the optimizations implemented by the V-FORCE client entail a consistent improvement of the users' QoE.

6. ACKNOWLEDGMENTS

Jeroen van der Hooft is funded by grant of the Agency for Innovation by Science and Technology in Flanders (IWT). The research was performed partially within the iMinds V-FORCE (Video: 4K Composition and Efficient streaming) project under IWT grant agreement no. 130655. This work was partly funded by FLAMINGO, a Network of Excellence project (318488) supported by the European Commission under its Seventh Framework Programme.

7. REFERENCES

[1] S. Akhshabi, S. Narayanaswamy, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptive video players over http. *Image Communication*, 2012.

[2] W. Cherif, Y. Fablet, E. Nassor, J. Taquet, and Y. Fujimori. Dash fast start using http/2. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2015.

[3] CONVIVA. 2015 viewer experience report. http://www.conviva.com/vxr-home/vxr2015/.

[4] H. Egilmez, S. Civanlar, and A. Tekalp. An optimization framework for qos-enabled adaptive video streaming over openflow networks. *IEEE Transactions on Multimedia*, 2013.

[5] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. Towards network-wide qoe fairness using openflow-assisted adaptive video streaming. In *Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking*, 2013.

[6] R. Houdaille and S. Gouache. Shaping http adaptive streams for a better user experience. In *Proceedings of the 3rd Multimedia Systems Conference*, 2012.

[7] R. Huysegems, J. van der Hooft, T. Bostoen, P. Rondao Alface, S. Petrangeli, T. Wauters, and F. De Turck. Http/2-based methods to improve the live experience of adaptive streaming. In *Proceedings of the 23rd ACM International Conference on Multimedia*, 2015.

[8] S. Petrangeli, J. Famaey, M. Claeys, S. Latré, and F. De Turck. Qoe-driven rate adaptation heuristic for fair adaptive video streaming. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 2015.

[9] S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck. Software-defined network-based prioritization to avoid video freezes in http adaptive streaming. *International Journal of Network Management (IJNM)*, 2016.

[10] S. Wei and V. Swaminathan. Low latency live video streaming over http 2.0. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, 2014.