View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Ghent University Academic Bibliography

An HTTP/2 Push-Based Approach for SVC Adaptive Streaming

Jeroen van der Hooft†, Stefano Petrangeli†, Niels Bouten†, Tim Wauters†, Rafael Huysegems§, Tom Bostoen§ and Filip De Turck†
† Ghent University - iMinds, Department of Information Technology, Technologiepark 15, B-9052 Ghent, Belgium
§ Alcatel Lucent - Bell Labs, Copernicuslaan 50, B-2018 Antwerp, Belgium
E-mail: jeroen.vanderhooft@intec.ugent.be

***Abstract*—HTTP Adaptive Streaming (HAS) is the de facto standard for over-the-top video streaming. In HAS, video content is encoded at multiple quality levels and temporally divided into multiple segments. The client can select the quality level for every video segment, allowing smoother playback and a better Quality of Experience (QoE). Although results are promising, current solutions often suffer from high round-trip time (RTT) cycles in mobile networks. This is especially true for scalable video coding (SVC), where multiple requests are required to retrieve a single video segment. Meanwhile, the IETF has standardized the HTTP/2 protocol since February 2015, providing new features that allow a reduction of the page load time in web browsing. In this paper, we propose a novel approach based on HTTP/2's server push feature to actively push the base layer of live, SVC-encoded content from server to client. This allows to eliminate one RTT cycle for every video segment, which has a significant impact on the user's QoE. Evaluating the proposed approach, we show that compared with HTTP/1.1, an improvement of 65.42% can be achieved for the average video quality in high-RTT networks. Compared to an AVC-based solution, the freeze frequency and duration are reduced by 54.55% and 53.06% respectively, while the loss in video quality is limited to 4.51%. Since playout freezes should be avoided at the cost of a lower video quality, we conclude that the proposed approach beneficially impacts the user's QoE.**

I. Introduction

Over the last years, delivery of multimedia content has become more prominent than ever. Recent studies show that more than half of the Internet traffic is generated by video streaming applications [1]. To meet increasing user requirements, the concept of HTTP Adaptive Streaming (HAS) has recently been introduced. As shown in Figure 1, video content is encoded at different quality levels and temporally divided into multiple segments with a typical length of 1 to 10 seconds. An HAS client can request these video segments in a dynamic way, changing the quality level of the requested segments whenever required. To this end, the client is equipped with a rate adaptation heuristic that selects the best quality level based on criteria such as the perceived bandwidth and the current buffer filling. The goal of this heuristic is to optimize the user's Quality of Experience (QoE), which depends among others on the average video quality, the frequency of quality changes and the occurrence of video freezes [2]. The client stores the incoming segments in a video buffer, before decoding the sequence in linear order and playing them out on the user's device. This approach offers a number of advantages. For

978-1-5090-0223-8/16/$31.00 © 2016 IEEE

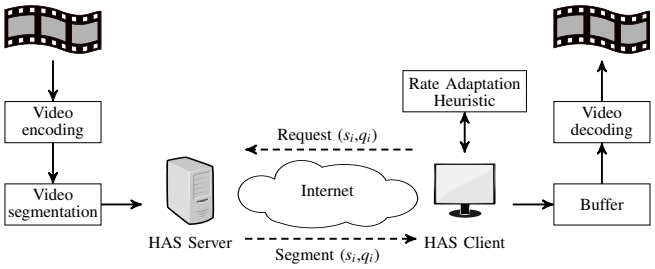


Figure 1: The concept of HTTP Adaptive Streaming.

the provider, video delivery is cheaper because no dedicated network elements are required. Better scalability is guaranteed, since quality selection is performed by clients in a distributed way. For the user, a smoother playback experience is generally perceived, because the client can adapt the requested bit rate to network conditions: when the available bandwidth suddenly drops, for instance, the client can select a lower quality level in order to prevent buffer starvation and playout freezes.

The encoding process of HAS solutions is often based on the H.264/AVC codec, which requires the server to store multiple independent representations of the same video. This generally results in storage overhead, increased bandwidth requirements and reduced caching efficiency. As suggested by Sánchez et al., the adoption of scalable video coding (SVC) in HAS offers a solution to this problem [3]. In SVC, redundancy is reduced by letting each quality level depend on the previous one. In the encoding process, lower layers are retrieved from a high-quality video bitstream by lowering the spatial or temporal resolution, the video quality signal or a combination thereof. Starting from the lowest quality level, called the base layer, the client can decode higher quality levels in combination with the lower layers. Not only does SVC provide an effective means to reduce content redundancy, it also results in a lower chance of buffer starvation. Indeed, since enhancement layers are downloaded one by one, the client can react faster to sudden changes of the available bandwidth. This is extremely important in HAS, where playout freezes should be avoided at the cost of a lower video quality [4]. A strong disadvantage to SVC however is that it introduces an encoding overhead of about 10% per layer, which increases the total bit rate of the content stream [5]. Furthermore, since multiple requests are required to retrieve a single video segment, SVC-based

solutions are more susceptible to high round-trip times (RTT). This problem mainly arises in mobile networks, where the RTT can vary from 33 to 857 ms, depending on the network carrier and the type of connection [6]. In this paper, we therefore propose an effective means to eliminate RTT cycles in SVC-based HAS, using the push feature of the recently standardized HTTP/2 protocol [7]. This approach allows to significantly reduce the freeze time in high-RTT networks compared to AVC-based solutions, while limiting the loss in quality by actively pushing content from server to client. Furthermore, we discuss the modifications required for two existing SVC-based rate adaptation heuristics, and present detailed experimental results to characterize the gain of the proposed approach compared to state-of-the-art HAS over HTTP/1.1.

The remainder of this paper is structured as follows. Section II gives an overview of related work, focusing both on HAS and HTTP/2. The proposed approach and used heuristics are presented in Section III. An evaluation is provided in Section IV, showing the most relevant results. In Section V, finally, conclusions are drawn and future work is discussed.

II. Related Work

A. HTTP Adaptive Streaming

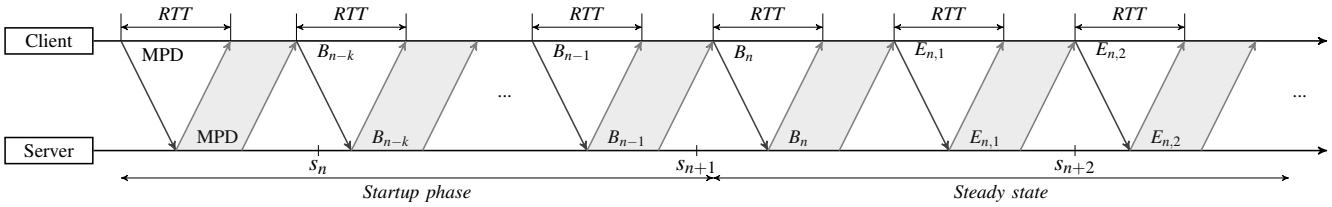
To improve the user's QoE for HAS services, both client-based, network-based and server-based solutions exist [4]. Well-known commercial rate adaptation heuristics are Microsoft's IIS Smooth Streaming (MSS), Apple's HTTP Live Streaming (HLS) and Adobe's HTTP Dynamic Streaming (HDS). As most of these implementations tend to use the same architecture, the Motion Picture Expert Group (MPEG) proposed Dynamic Adaptive Streaming over HTTP (DASH), a standard that defines the interfaces and protocol data for adaptive video streaming over HTTP [8]. The heuristics are however still implementation specific. In literature, Benno et al. propose a more robust rate adaptation heuristic for wireless live streaming [9]. By averaging the measured bandwidth over a sliding window, fluctuations are smoothed, allowing the client to select a quality level that is sustainable and avoids oscillations. Claeys et al. propose to use reinforcement learning, introducing a Q-learning algorithm in the adaptation heuristic that allows to outperform certain deterministic algorithms such as MSS [10]. Focusing on high-RTT networks, Bouten et al. propose to use pipelined and parallel download scheduling to reduce the negative impact of the RTT on the user's QoE [11]. In the suggested approach, the client issues multiple GET requests using HTTP pipelining or parallel TCP connections, eliminating the idle time between two successive downloads. However, since multiple segments are being transferred at the same time, the approach is more vulnerable to network congestion and buffer starvation when sudden drops in the available bandwidth occur. A large number of other client-based solutions exist, but we refer to the survey by Seufert et al. for a more elaborate view on the matter [4].

In network-based solutions, clients attempt to reach a globally optimal QoE, instead of optimizing their own QoE. Petrangeli et al. suggest an approach in which each client learns to select the most appropriate quality level, maximizing a reward based both on its own QoE and on the QoE perceived by other clients [12]. To this end, a coordination proxy estimates all perceived rewards and generates a global signal that is sent periodically to all clients. Without explicit communication among agents, the algorithm is able to outperform both MSS and the algorithm proposed by Claeys et al. in a multi-client scenario. Bouten et al. propose to introduce intelligence in the network that steers the client's local quality decisions, by modifying the announced adaptation set [13], [14]. Using HAS aware network elements, video quality levels are assigned to specific clients subject to their respective subscription terms, improving fairness among users while still allowing them to adapt to dynamic network changes [15].

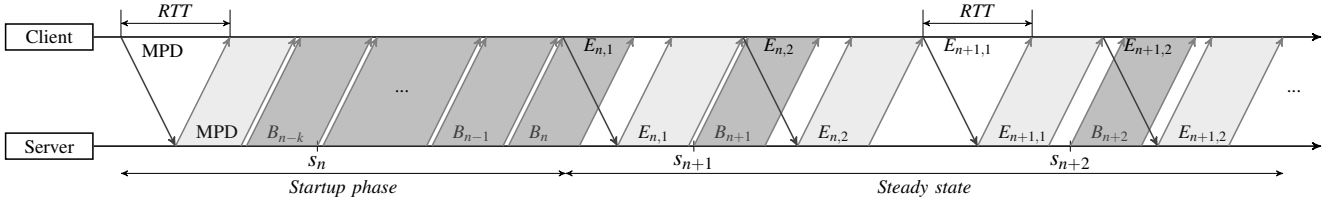
Server-side solutions typically focus on new encoding schemes for HAS [4]. The H.264/AVC codec is most widely used for video streaming, although it requires the server and intermediate caches to store multiple representations of the same video. Huysegems et al. discuss the theoretic advantages of SVC in HAS, being a smoother play-out in networks with high variability and a significant reduction in storage and bandwidth requirements [16]. The authors also identify two important challenges for SVC, such as a penalty in the total bitrate for encoding SVC and an increased vulnerability to high RTTs. Sánchez et al. discuss the benefits of SVC in HAS, in terms of web caching and saved uplink bandwidth [3], [17]. Furthermore, a scheduling algorithm for live HAS delivery is proposed. An initial comparison of SVC and AVC is carried out, focusing on the observed camera-to-display delay in live video streaming. In more recent work, Bouten et al. propose to use Differentiated Services (DiffServ) in the IP network to give priority to the base-layer segments in SVC-based HAS [18]. As a result, the clients are more robust to video freezes, even when the total buffer size is decreased to two seconds in order to reduce the end-to-end delay for live video streaming. In contrast to these works, which all focus on content delivery over HTTP/1.1, we propose a new approach in which HTTP/2 is used to push the base layer of new video segments in SVC-based live streaming. As will be shown in Section IV, this approach provides an effective means to eliminate RTT cycles, significantly improving the user's QoE in high-RTT networks.

B. HTTP/2 for Multimedia Delivery

Early 2012, the IETF httpbis working group started the standardization of HTTP/2 to address a number of deficiencies in HTTP/1.1 [19], [20]. The new HTTP/2 standard was published as an IETF RFC in February 2015, and is now supported by major browsers such as Google Chrome, Firefox and Internet Explorer [7]. The main focus of this standard is to reduce the latency in web delivery, using three new features that provide the possibility to terminate the transmission of certain content, prioritize content and push content from server to client. The first draft for this protocol was based on SPDY, an open networking protocol developed primarily by Google [21]. Using this protocol, an average reduction of up to 64% was observed for the page load time.



(a) Adaptive streaming using traditional HTTP/1.1.



(b) Adaptive streaming using HTTP/2's server push.

Figure 2: An example scenario for live SVC video streaming over HTTP/1.1 and HTTP/2, showing both the startup phase and steady state. B_k and $E_{k,j}$ denote the base layer and jth enhancement layer of segment k respectively, while s_k denotes the release of segment k at server-side. Light gray filling indicates a pulled resource, dark gray filling a pushed resource.

Mueller et al. were the first to evaluate the performance of adaptive streaming over SPDY [22]. The existing HTTP/1.1 layer is replaced by SPDY, without any further modifications to the client or server. The authors show that the gains obtained by using header compression, a persistent connection and pipelining are almost completely canceled out by the losses due to the required SSL and framing overhead. Wei et al. explore how new HTTP/2 features can be used to improve HAS [23]. By reducing the segment duration from five seconds to one second, they manage to reduce the camera-to-display delay from about 16 to 4 seconds. An increased number of GET requests is avoided by using HTTP/2's server push to push k segments after each request. The main disadvantage of this approach is that when a client wants to switch to a new video quality, the push stream for the old quality is in competition with the stream for the new quality. This results in bandwidth overhead and causes an increased switching delay for the client. In later work, the authors show that this switching delay is about two segment durations and independent of the value of k, while the introduced bandwidth overhead heavily depends on this value [24]. Moreover, HTTP/2 is used to push audio upon receiving a request for the associated video segments.

In previous work, we proposed a large number of HTTP/2-based techniques that can improve the QoE in HAS, using stream prioritization, stream termination and server push [25]. Each technique has its own advantages, such as a higher bandwidth utilization and a gain of multiple RTT cycles in the client startup phase. We thus proposed a *full-push* approach, in which several techniques are combined. A preliminary evaluation showed promising results, such as a lower camera-to-display delay and startup time in high-RTT networks. A disadvantage of this approach is that, if sudden drops in the available bandwidth occur, network congestion is more likely to occur when segments are being pushed at a high quality. This is why, in this paper, we focus on a new approach for SVC video, in which only the base layer is pushed from server to client.

III. Proposed Approach

In Section III-A, we explain how the push-based approach reduces the negative impact of high RTTs on the QoE for HAS. To this end, two existing rate adaptation heuristics for SVC-based HAS are briefly discussed. These heuristics were designed with the possibilities of HTTP/1.1 in mind, and are thus unable to take advantage of the new features offered by HTTP/2. In Section III-B, we therefore propose two novel heuristics that combine the benefits of these existing adaptation heuristics and HTTP/2's server push.

A. Base Layer Push

In HAS, a video streaming session starts with a startup phase, in which the client sends a request for the video's media presentation description (MPD) or manifest file. This file contains information regarding the video segments, such as the duration, resolution and available quality representations. In live video, it also contains information regarding the timing of the video streaming and the segments already available on the server. 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. Once the buffer filling is sufficiently high - usually when a certain threshold is exceeded - further decisions regarding the video quality are made by the rate adaptation heuristic. The main drawback of using SVC in HAS is that multiple RTT cycles are lost to enhance the video quality, which has a significant impact on the bandwidth utilization and the startup time in high RTT networks. This behavior is illustrated in Figure 2a, where a live streaming session is shown. Once segment s_n is released at server side, the client can request the base layer B_n and consecutive enhancement layers $E_{n,1},\ldots,E_{n,m}$. Although the use of SVC in HAS provides a smoother playout and strongly reduces the storage overhead, the approach does not perform well when the RTT is relatively high compared to the segment duration.

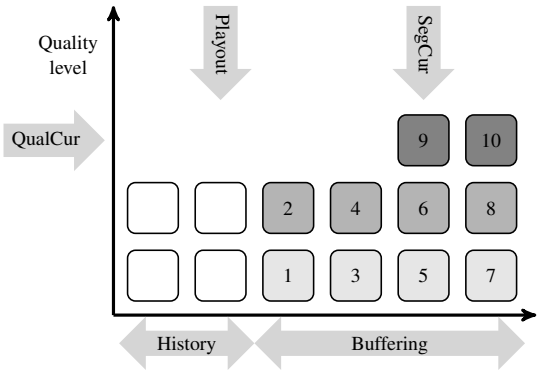


Figure 3: Illustration of the SVC-Cursor heuristic.

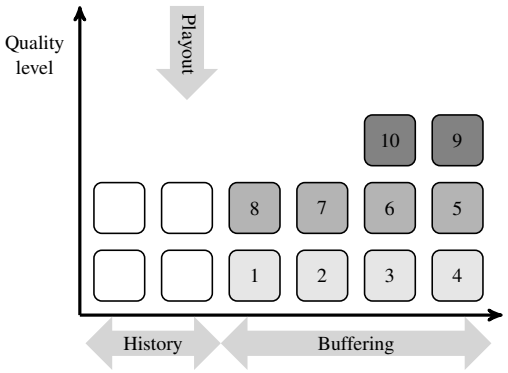


Figure 4: Illustration of the SVC-BackFilling heuristic.

In the push-based approach, the server first enters a startup phase in which the last k base layer segments $B_{n-k},...,B_{n-1}$ are pushed to the client as soon as an MPD request is received, in order to ramp up the buffer at client-side. This requires that the server is aware of the relationship between the different HAS objects, and that the client can indicate its temporal buffer size in the MPD request. In this way, the client's startup time is reduced by one RTT and, depending on which rate adaptation heuristic is used, the total buffer ramp up time is reduced by at most k times the RTT. Once the MPD is returned and the first k segments are pushed, the server enters a steady state, periodically pushing a newly released base layer to the client. As illustrated in Figure 2b, this entails a gain of one RTT cycle in the reception of every video segment. In high RTT networks, this approach has the potential to achieve a higher bandwidth utilization, and thus to provide the user with a higher average video quality. The main advantage of this approach, compared to our full-push approach for AVC video in [25], is that network congestion is less likely to occur: only base layers are being pushed, representing the minimum amount of information every client needs to stream the video.

Note that there are two ways for the client to request the enhancement layers: either the same HTTP/2 connection can be reused, or a new, parallel connection can be established. A disadvantage of the former approach is that, when an enhancement layer is requested just before a base layer is pushed, this enhancement layer is first transferred completely. In this way, the base layer will arrive significantly later, making the client potentially more susceptible to buffer starvation. Using the latter approach, the two layers will be transferred in parallel, allowing the base layer to be pushed immediately to the client. When the request for an enhancement layer however arrives immediately after a base layer has been released, the base layer will arrive slightly later, since the two connections compete for the available bandwidth. In Section IV, we will show that these two approaches lead to very similar results.

B. Rate Adaptation Heuristics

In this paper, we focus on two rate adaptation heuristics: SVC-Cursor and SVC-Backfilling [11], [26]. These heuristics are briefly discussed below, along with the changes required to realize the push-based approach presented in Section III-A.

1) SVC-Cursor: In HAS, the average video quality should be as high as possible, while buffer starvation and quality switches should be avoided. The SVC-Cursor heuristic, proposed by Bouten et al., attempts to achieve just that [11]. Two different cursors are used: a segment cursor, which defines the next segment under consideration, and a quality cursor, which defines the target quality level. The segment cursor moves to the next segment when either all quality levels up to the quality cursor are downloaded for the current segment, or the considered enhancement layer cannot be downloaded in time. In the latter case, the quality cursor is immediately decreased and the improvement timeout is reset. This cursor is incremented only when all lower layers of every segment in the buffer are downloaded and the improvement timer has expired. In this case, the segment cursor is moved based on the estimations of the arrival times of the enhancement layers and their playout times, evaluated from right to left. As such, the cursor is set to the segment index i for which the estimated bandwidth allows to download all enhancement layers for the segments $i, i+1, i+2...$ in time for their respective playout, and the improvement timer is reset. The enhancement layers are then downloaded from left to right, as illustrated in Figure 3. In this example scenario, the quality cursor was increased after the arrival of the enhancement layer with index 8.

To use this heuristic with the proposed push-based approach, a number of changes are required. First, the minimum quality level for the quality cursor is changed to the first enhancement layer, so that base layer segments are never pulled by the client. Furthermore, the client only requests enhancement layer segments when at least a configurable number of base layer segments are available. In this way, pulled enhancement layer segments are not preventing base layer segments from arriving, possibly leading to a higher number of playout freezes. Second, enhancement layers are not requested for the first segment in the buffer, since chances are high that this segment will be played out before the enhancement layer arrives. Third, the bandwidth estimation process is modified to take into account the perceived throughput of both the base layers and the enhancement layers. Since no information on the RTT is available for the pushed base layer segments on the application layer, this leads to slightly different values than would be the case for a fully pull-based approach.

	AVC		SVC	
	Avg. [kb/s]	Max. [kb/s]	Avg. [kb/s]	Max. [kb/s]
Quality 0	336	866	309	849
Quality 1	1086	3082	1240	3473
Quality 2	1984	5702	2529	7224

Table I: Cumulative bit rates of the quality levels.

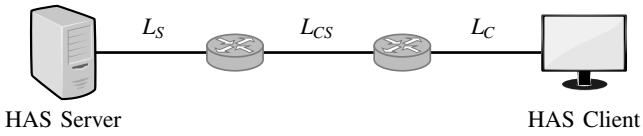


Figure 5: Experimental Mininet setup.

2) SVC-Backfilling: In bandwidth-based heuristics, the next quality level is based on an estimation of the available bandwidth. In mobile, highly variable networks however, a reliable estimation cannot be performed. As such, the client is more likely to expose itself to buffer starvation and thus to playout freezes. Using SVC mitigates this problem, since the client can play out the next segment as long as the base layer is available. In some cases, however, even an SVC client can experience video freezes if the bandwidth suddenly drops. One possible solution is to use purely buffer-based rate adaptation heuristics, in which only the buffer filling is used to decide upon the next segment and quality to download. One such example is the SVC Backfilling heuristic, proposed by Petrangeli et al. [26]. In this heuristic, the client always checks whether or not all base layers have been downloaded. If this is not the case, the base layer of the segment with the earliest playout time is first downloaded. Once all base layers are available, the heuristic starts to upgrade the segments with the latest playout time, as shown in Figure 4. In this way, the client makes sure that all base layers are downloaded before starting to upgrade the quality. Furthermore, since upgrading the enhancement layers is done from right to left, more gradual quality switches are expected. To use this heuristic with the proposed push-based approach, the base layer logic is eliminated, simply downloading the enhancement layers from right to left. Again, a configurable minimum required number of base layer segments is used to avoid competing base layer and enhancement layer segments.

IV. Evaluation and Discussion

To illustrate the gains of the suggested approach, we evaluated the proposed heuristics for different network conditions and RTT values. The experimental setup is presented below, followed by an overview of considered evaluation metrics and a detailed overview of the obtained results.

A. Experimental Setup

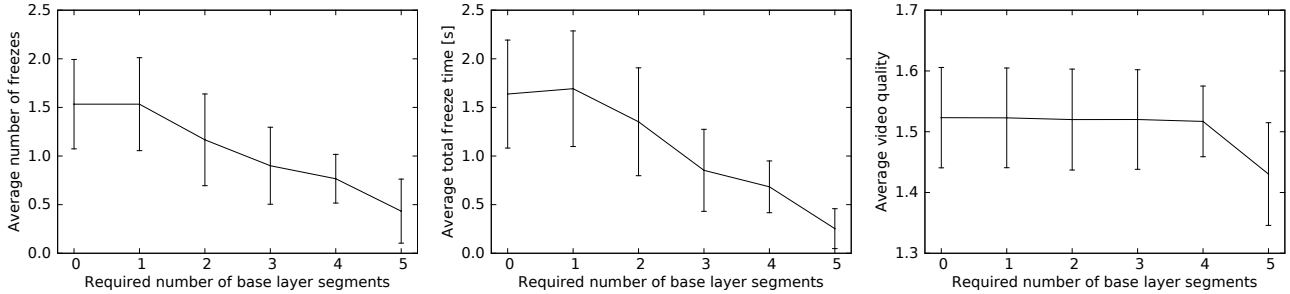
The considered video sequence in our experiments has a total length of 300 seconds, with a frame rate of 24 FPS. The video is encoded at a variable bit rate, both for H.264/AVC and H.264/SVC. It consists of 300 segments, each one second of length and available in three quality representations. Table I shows the average and maximum cumulative bit rates of each quality level. The differences in bit rates between AVC and SVC of the base layer are due to optimizations of the encoder, while the differences of higher layers are a consequence of the encoding overhead of SVC. For this video, the average overhead is 14.2% and 27.5% for layers 1 and 2 respectively.

To evaluate the proposed approaches, the network topology in Figure 5 is emulated using the MiniNet framework[1]. It consists of a single client, streaming live video from a dedicated HAS server. 30 episodes of the video are streamed for every evaluated configuration, using a different bandwidth pattern for every episode on link L_{CS}. These patterns are extracted from an open-source dataset, collected by Riiser et al. on a real 3G/HSDPA network [27]. However, a lower threshold of 50 kb/s is used to guarantee correct traffic shaping with tc. The average available bandwidth in the traces is 2354 kb/s, with a standard deviation of 1393 kb/s. The available bandwidth on links L_S and L_C is fixed at 3.2 Mb/s. This setup corresponds to a realistic scenario, and allows a fair comparison of the proposed push-based approach with solutions over HTTP/1.1.

As for the HAS server, its implementation is based on the Jetty web server[2]. Jetty's HTTP/2 component allows to define a push-based strategy, which describes all resources that need to be pushed along with the requested resource. Such a strategy is ideal for web-based content, where the required JavaScript and CSS files, images and other content can immediately be pushed. However, since we target a livestream scenario, 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 livestream request, passing along the maximum buffer size. When this request corresponds to a new session, the server starts a push thread that pushes the k last released video segments at the lowest quality immediately. In this way, the client can quickly ramp up its buffer without overloading the network. Note that no real-time footage is captured at server-side; segments are simply being released periodically.

The HAS client is implemented on top of the libdash library[3], the official reference software of the ISO/IEC MPEG-DASH standard. To make use of HTTP/2's server push, a number of changes were made. First, an HTTP/2-based connection was added to enable the reception of pushed segments. To this end, the nghttp2 library[4] was used to set up an HTTP/2 connection over SSL. Second, support for SVC and the transparent base layer push was provided by extending the DASH Receiver, allowing heuristics to select both a new segment index and video quality. Third, the bandwidth estimation process was adapted to take into account the presence of pushed base layer segments in the network. The client is equipped with three rate adaptation heuristics: SVC-Cursor, SVC-Backfilling and the AVC-based FINEAS

[1] http://mininet.org/
[2] https://webtide.com/
[3] https://github.com/bitmovin/libdash/
[4] https://nghttp2.org/



(a) Average number of freezes (b) Average total freeze time (c) Average video quality

Figure 6: Impact of the required number of base layer segments on (a) the average video quality, (b) the number of freezes and (c) the total freeze time, for SVC-Backfilling with a shared HTTP/2 connection and a negligible RTT.

algorithm by Petrangeli et al. [28]. In recent work, the authors showed that the latter outperforms well-known heuristics such as MSS. Note that, since a single-client scenario is considered, the proposed in-network computation has not been explicitly used. To limit the camera-to-display delay for the livestream, the client initially starts with a temporal buffer size of five seconds, or five video segments. When a freeze occurs, the buffer size is increased in order to hold all released segments. Once the temporal buffer size exceeds the maximum buffer size, set to twenty seconds in our setup, segments are skipped in order to keep up with the live signal. In our experiments it turned out, however, that the client is capable of following the live signal very closely, never exceeding this threshold.

B. Evaluation Metrics

The following evaluation metrics are considered to compare the performance of the proposed HTTP/2-based approach with HTTP/1.1 and traditional AVC-based HAS. First, the average video quality, expressed as a number between 0 (corresponding to the base layer) and 2 (corresponding to the second enhancement layer). Second, the frequency and total duration of playout freezes. Third, the initial startup delay, defined as the time between requesting the livestream at client-side and the playout of the first video segment. To evaluate the impact of the proposed approaches on these evaluation metrics, the considered video trace is streamed thirty times for every configuration. Results are therefore shown using the observed averages and the 95% confidence intervals.

C. Base Layer Segments

In this section, the optimal value for the minimum required number of base layer segments is determined. This parameter indicates the number of base layer segments that should be available in the buffer, before enhancement layer segments can be requested. For a temporal buffer size of five seconds, containing five video segments, possible values are thus 0 up to 5. Figure 6 shows the impact of this parameter value on (a) the number of freezes, (b) the total freeze time and (c) the average video quality, for the SVC-Backfilling heuristic over a shared HTTP/2 connection with a negligible RTT. For larger values of the parameter, a significant reduction is observed for the frequency and length of freezes (Wilcoxon signed-rank test, $p < 0.05$). This behavior is explained by the fact that, when more base layer segments are available before enhancement layers are downloaded, there is a reduced chance of buffer starvation. The average video quality is more or less similar for values of 0 to 4, but a reduction is observed for a value of 5 (Wilcoxon signed-rank test, $p < 0.05$). This behavior is explained as follows. When the server initially starts pushing base layer segments, the first segment is immediately played out by the client. The buffer will eventually contain between 4 and 5 video segments, and as such, the client will often find itself waiting for a new segment to be pushed. This leads to a lower bandwidth utilization, and thus to a lower average video quality. Similar trends were observed for the SVC Cursor heuristic, for two parallel connections and for larger RTTs. As such, a value of 4 was selected for the evaluations below.

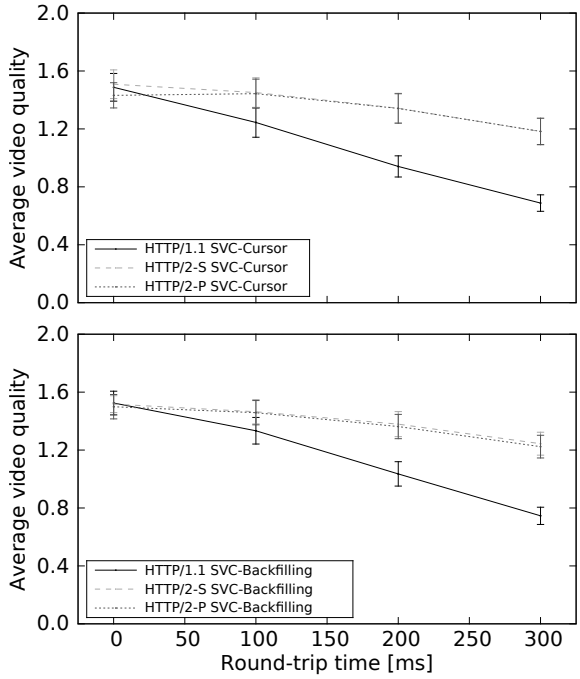


Figure 7: Impact of the RTT on the average video quality, for HTTP/1.1, for a shared HTTP/2 connection (HTTP/2-S) and two parallel connections (HTTP/2-P). The top row shows results for SVC-Cursor, the bottom row for SVC-Backfilling.

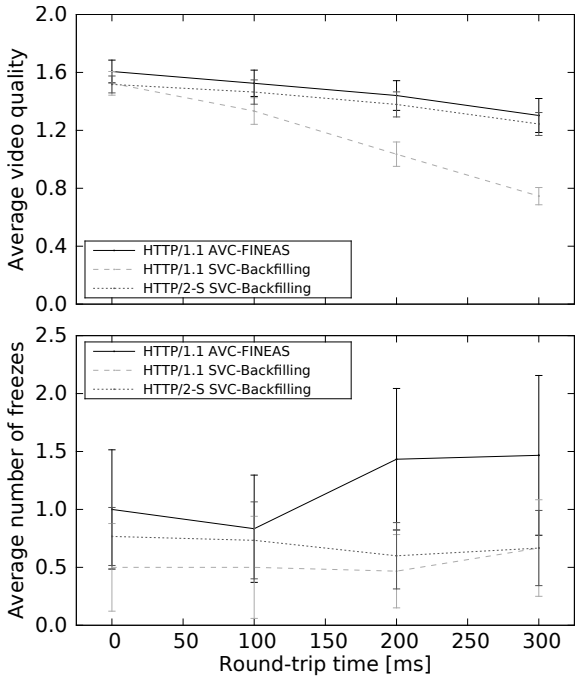


Figure 8: Impact of an increasing RTT for AVC-FINEAS over HTTP/1.1, SVC-Backfilling over HTTP/1.1 and SVC-Backfilling over HTTP/2. The top row shows the average video quality, the bottom row the average number of freezes.

D. Evaluation Results

Figure 7 shows the average video quality achieved for the two SVC heuristics, using the benchmark HTTP/1.1, HTTP/2 with a single shared connection and HTTP/2 with two parallel connections. For the SVC-Backfilling heuristic over HTTP/1.1, the average video quality level is reduced from 1.483 for a negligible RTT to 0.746 for an RTT of 300 ms (-49.70%). This is the consequence of lost RTT cycles in SVC, strongly reducing the bandwidth utilization. Using HTTP/2-S and HTTP/2-P, however, the average video quality is 1.234 for an RTT of 300 ms ($+65.42\%$). Bandwidth utilization is significantly more efficient in this case: one RTT cycle is gained for every video segment, and, while a request for an enhancement layer is still on-the-fly, a base layer segment can be pushed in the meantime. Even for an RTT of 100 ms, this results in an average gain of 9.58%. As for HTTP/2-S and HTTP/2-P, no significant difference between the video quality is observed. This is because overall, the average bandwidth throughput for the two approaches is in fact the same. Similar results for the proposed approach are obtained for SVC-Cursor, showing the general applicability of the proposed solution.

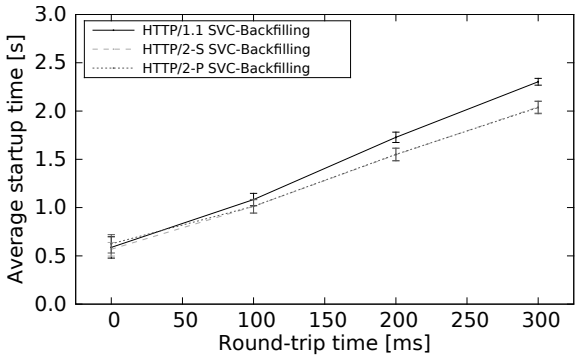


Figure 9: Impact of an increasing RTT on the client's startup time, for HTTP/1.1, for a shared HTTP/2 connection (HTTP/2-S) and two parallel connections (HTTP/2-P).

Figure 8 shows results for AVC-FINEAS over HTTP/1.1 with SVC-Backfilling over HTTP/1.1 and HTTP/2-S. While the decreasing trend for the average video quality for AVC-FINEAS and SVC-Backfilling over HTTP/2 is similar, the former is on average 4.82% higher. This is a consequence of the additional overhead introduced in SVC-encoded solutions. Comparing AVC with SVC, however, a higher frequency of playout freezes is observed (Wilcoxon signed-rank test, $p < 0.05$). This is because, on average, the download time for the complete AVC segments is higher than for the SVC-encoded layered segments. When a low buffer filling is observed, the SVC-based client can thus ramp up the buffer more quickly, resulting in a lower chance of buffer starvation.

Figure 9, finally, shows the average startup delay of the client, which is independent of the applied heuristic. This delay is similar for the different approaches, when the network's RTT is negligible; in this case, there is no observable difference between the pulling or pushing of video segments. For higher RTTs however, an increasing difference of the startup time is observed. This difference corresponds to one RTT cycle, which is gained by immediately pushing the first base layer along with the MPD. This means that the HTTP/2-based client is able to start the playout of new video streams faster in high-RTT networks, which again improves the user's QoE.

To concretize the gain of the proposed approach, results are summarized in Table II, for an RTT of 300 ms. Comparing results with the AVC-based approach, the freeze frequency and duration are reduced by 54.55% and 53.06% respectively, while the loss in video quality is limited to 4.51%. Since playout freezes should be avoided at the cost of a lower video quality, we conclude that our approach beneficially impacts the user's QoE [4].

HTTP	Heuristic	Video quality	Freeze frequency	Freeze time [s]	Startup delay [s]
HTTP/1.1	AVC-FINEAS	1.303 ± 0.117	1.467 ± 0.690	1.144 ± 0.736	2.332 ± 0.040
HTTP/1.1	SVC-Backfilling	0.746 ± 0.059	0.667 ± 0.417	0.337 ± 0.233	2.304 ± 0.035
HTTP/2-S	SVC-Backfilling	1.244 ± 0.079	0.667 ± 0.324	0.537 ± 0.302	2.043 ± 0.062

Table II: Performance summary for the different heuristics, comparing results for an RTT of 300 ms. The average values are reported, together with the 95% confidence intervals.

V. Conclusions

In this paper, we proposed a new approach for SVC-based adaptive streaming that uses the server push feature provided by the recently standardized HTTP/2 protocol. We introduced two novel heuristics, combining the benefits of HTTP/2's server push with those of two state-of-the-art rate adaptation heuristics for HTTP/1.1. Evaluating the obtained results through emulation, we showed that compared to HTTP/1.1-based SVC, the average video quality can be improved with 9.58% for an RTT of 100 ms. For an RTT of 300 ms, common in mobile, high-RTT networks, the average gain is even up to 65.42%. Compared to the AVC-based FINEAS heuristic, the freeze frequency and duration are reduced by 54.55% and 53.06% respectively, while the loss in video quality is limited to 4.51%. Furthermore, pushing the first base layer segments alongside the requested MPD, the client's startup time is reduced by one RTT cycle.

Future work will focus on more possibilities to improve the user's QoE using the new HTTP/2 features, both in AVC- and SVC-based adaptive streaming. One interesting research path includes a multi-client scenario, where the focus will be both on maximizing the average QoE and providing fairness among clients.

Acknowledgments

Jeroen van der Hooft and Niels Bouten are 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.

References

[1] Sandvine Incorporated, "Global Internet Phenomena Report, 1H 2014," 2014.

[2] R. Mok, E. Chan, and R. Chang, "Measuring the Quality of Experience of HTTP Video Streaming," in *2011 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2011, pp. 485–492.

[3] Y. Sánchez de la Fuente, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, and Y. Le Louédec, "iDASH: Improved Dynamic Adaptive Streaming over HTTP Using Scalable Video Coding," in *Proceedings of the 2nd Annual ACM Conference on Multimedia Systems*, ser. MMSys '11. ACM, 2011, pp. 257–264.

[4] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hossfeld, and P. Tran-Gia, "A Survey on Quality of Experience of HTTP Adaptive Streaming," *Communications Surveys Tutorials, IEEE*, vol. 17, no. 1, pp. 469–492, 2015.

[5] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard," *Transactions on Circuits and Systems for Video Technology, IEEE*, vol. 17, no. 9, pp. 1103–1120, 2007.

[6] OpenSignal, "IConnect 4G Coverage Maps," 2014. [Online]. Available: http://opensignal.com/networks/usa/iconnect-4g-coverage

[7] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2," RFC Editor, Tech. Rep. Internet-Draft, 2015. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-httpbis-http2/

[8] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP: Standards and Design Principles," in *Proceedings of the 2nd Annual ACM Conference on Multimedia Systems*, 2011, pp. 133–144.

[9] S. Benno, A. Beck, J. Esteban, L. Wu, and R. Miller, "WiLo: A Rate Determination Algorithm for HAS Video in Wireless Networks and Low-Delay Applications," in *Globecom Workshops (GC Wkshps), 2013 IEEE*, 2013, pp. 512–518.

[10] M. Claeys, S. Latré, J. Famaey, and F. De Turck, "Design and Evaluation of a Self-Learning HTTP Adaptive Video Streaming Client," *Communications Letters, IEEE*, vol. 18, no. 4, pp. 716–719, 2014.

[11] N. Bouten, S. Latré, J. Famaey, F. De Turck, and W. Van Leekwijck, "Minimizing the Impact of Delay on Live SVC-Based HTTP Adaptive Streaming Services," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2013, pp. 1399–1404.

[12] S. Petrangeli, M. Claeys, S. Latré, J. Famaey, and F. De Turck, "A Multi-Agent Q-Learning-Based Framework for Achieving Fairness in HTTP Adaptive Streaming," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, 2014, pp. 1–9.

[13] N. Bouten, J. Famaey, S. Latré, R. Huysegems, B. Vleeschauwer, W. Leekwijck, and F. Turck, "QoE Optimization Through In-Network Quality Adaptation for HTTP Adaptive Streaming," in *Network and Service Management (CNSM), 2012 8th International Conference and 2012 Workshop on Systems Virtualization Management (SVM)*, 2012, pp. 336–342.

[14] N. Bouten, S. Latré, J. Famaey, W. Van Leekwijck, and F. De Turck, "In-Network Quality Optimization for Adaptive Video Streaming Services," *Transactions on Multimedia, IEEE*, vol. 16, no. 8, pp. 2281–2293, 2014.

[15] N. Bouten, R. de Oliveira Schmidt, J. Famaey, S. Latré, A. Pras, and F. De Turck, "QoE-Driven In-Network Optimization for Adaptive Video Streaming Based on Packet Sampling Measurements," *Computer Networks*, 2015.

[16] R. Huysegems, B. De Vleeschauwer, T. Wu, and W. Van Leekwijck, "SVC-Based HTTP Adaptive Streaming," *Bell Labs Technical Journal*, vol. 16, no. 4, pp. 25–41, 2012.

[17] Y. Sánchez de la Fuente, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, and Y. Le Louédec, "Efficient HTTP-Based Streaming Using Scalable Video Coding," *Signal Processing: Image Communication*, vol. 27, no. 4, pp. 329–342, 2012.

[18] N. Bouten, M. Claeys, S. Latré, J. Famaey, W. Van Leekwijck, and F. De Turck, "Deadline-Based Approach for Improving Delivery of SVC-Based HTTP Adaptive Streaming Content," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, 2014.

[19] M. Belshe, R. Peon, M. Thomson, and A. Melnikov, "SPDY Protocol," *IETF*, 2012. [Online]. Available: https://tools.ietf.org/html/draft-ietf-httpbis-http2-00

[20] IETF, "Hypertext Transfer Protocol (httpbis)," *IETF*, 2012. [Online]. Available: https://datatracker.ietf.org/wg/httpbis/charter/

[21] Google, "SPDY: An Experimental Protocol for a Faster Web," Tech. Rep., 2009. [Online]. Available: http://www.chromium.org/spdy/spdy-whitepaper

[22] C. Mueller, S. Lederer, C. Timmerer, and H. Hellwagner, "Dynamic Adaptive Streaming over HTTP/2.0," in *2013 IEEE International Conference on Multimedia and Expo (ICME)*, 2013, pp. 1–6.

[23] S. Wei and V. Swaminathan, "Low Latency Live Video Streaming over HTTP 2.0," in *Proceedings of the 24th Network and Operating System Support on Digital Audio and Video Workshop*, ser. NOSSDAV '14. ACM, 2014, pp. 37:37–37:42.

[24] S. Wei and V. Swaminathan, "Cost Effective Video Streaming Using Server Push over HTTP 2.0," in *2014 IEEE 16th International Workshop on Multimedia Signal Processing (MMSP)*, 2014, pp. 1–5.

[25] R. Huysegems, J. van der Hooft, T. Bostoen, P. 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 Multimedia Conference*. ACM, 2015.

[26] S. Petrangeli, N. Bouten, M. Claeys, and F. De Turck, "Towards SVC-based Adaptive Streaming in Information Centric Networks," in *2015 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, 2015, pp. 1–6.

[27] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Video Streaming Using a Location-Based Bandwidth-Lookup Service for Bitrate Planning," *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 8, no. 3, pp. 24:1–24:19, 2012.

[28] 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, in Press.