J Netw Syst Manage (2018) 26:51–78
https://doi.org/10.1007/s10922-017-9407-2

An HTTP/2 Push-Based Approach for Low-Latency Live Streaming with Super-Short Segments

Jeroen van der Hooft[1] · Stefano Petrangeli[1] · Tim Wauters[1] · Rafael Huysegems[2] · Tom Bostoen[2] · Filip De Turck[1]

Received: 28 November 2015 / Revised: 22 February 2017 / Accepted: 25 February 2017 / Published online: 3 March 2017

Abstract Over the last years, streaming of multimedia content has become more prominent than ever. To meet increasing user requirements, the concept of HTTP Adaptive Streaming (HAS) has recently been introduced. In HAS, video content is temporally divided into multiple segments, each encoded at several quality levels. A rate adaptation heuristic selects the quality level for every segment, allowing the client to take into account the observed available bandwidth and the buffer filling level when deciding the most appropriate quality level for every new video segment. Despite the ability of HAS to deal with changing network conditions, a low average quality and a large camera-to-display delay are often observed in live streaming scenarios. In the meantime, the HTTP/2 protocol was standardized in February 2015, providing new features which target a reduction of the page loading time in web browsing. In this paper, we propose a novel push-based approach for HAS, in which HTTP/2's push feature is used to actively push segments from server to client. Using this approach with video segments with a sub-second duration, referred to as *super-short* segments, it is possible to reduce the startup time and end-to-end delay in HAS live streaming. Evaluation of the proposed approach, through emulation of a multi-client scenario with highly variable bandwidth and latency, shows that the startup time can be reduced with 31.2% compared to traditional solutions over HTTP/1.1 in mobile, high-latency networks. Furthermore, the end-to-end delay in live streaming scenarios can be reduced with 4 s, while providing the content at similar video quality.

✉ Jeroen van der Hooft
jeroen.vanderhooft@ugent.be

[1] Department of Information Technology, Ghent University - imec, Technologiepark 15, 9052 Ghent, Belgium

[2] Bell Labs, Nokia, Copernicuslaan 50, 2018 Antwerp, Belgium

Keywords HTTP adaptive streaming · Live video streaming · End-to-end delay · Quality of experience · HTTP/2 · Server push · Short segment duration

1 Introduction

Over the last years, delivery of multimedia content has become more prominent than ever. Particularly, video streaming applications are currently responsible for more than half of the Internet traffic [1]. To improve video streaming over the best-effort Internet, the concept of HTTP Adaptive Streaming (HAS) has recently been introduced. As shown in Fig. 1, video content is temporally divided into segments with a typical length of 1–10 s, each encoded at multiple quality levels. Video segments are requested by the HAS client, equipped with a rate adaptation heuristic to select the best quality level based on criteria such as the perceived bandwidth and the video player's buffer filling level. The ultimate objective of this heuristic is to optimize the Quality of Experience (QoE) perceived by the user, which depends among others on the average video quality, the frequency of quality changes and the occurrence of playout freezes [2]. The client decodes all incoming segments and uses these to create a continuous playout experience.

This approach comes with a number of advantages. For the over-the-top provider, video delivery is cheaper because the existing HTTP infrastructure for web surfing can be reused. Better scalability is guaranteed, since quality selection is performed by clients in a distributed way. The user generally perceives a smoother playback experience, because the client adapts the requested bit rate to network conditions; when the perceived bandwidth drops for instance, a lower quality level can be selected to prevent buffer starvation. Because of these advantages, major players such as Microsoft, Apple, Adobe and Netflix massively adopted the adaptive streaming paradigm and proposed their own rate adaptation heuristics. As most HAS solutions use the same architecture, the Motion Picture Expert Group (MPEG) proposed Dynamic Adaptive Streaming over HTTP (DASH), a standard which defines the interfaces and protocol data for adaptive streaming [3].

Despite the many advantages brought by HAS, several inefficiencies still have to be solved in order to improve the QoE perceived by the user in live video streaming

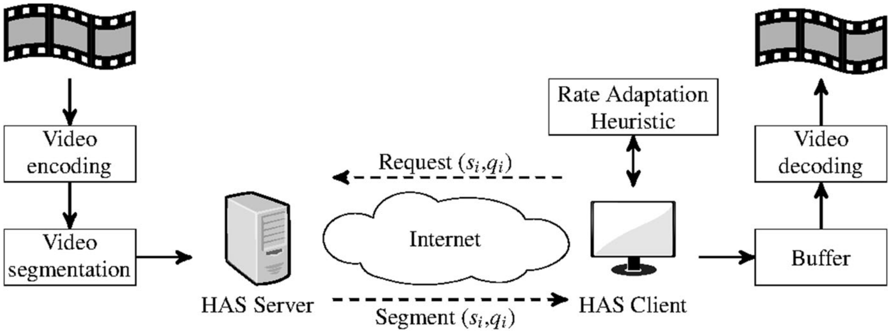


Fig. 1 The concept of HTTP adaptive streaming

scenarios. 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 in a live video scenario. In current HAS deployments however, this delay is in the order of tens of seconds. This is because encoding video in real-time is a computationally expensive and time consuming process, a large buffer at the client is generally used to prevent playout freezes and the server only sends a new video segment once a request is issued by the client. One possible way to lower this delay is to use segments with a sub-second duration, henceforth referred to as *super-short* segments: less time is required for encoding at the head-end and transport over the network [4], while a lower temporal buffer size can be used. The rate adaptation heuristic can also adapt faster to changing network conditions, preventing buffer starvation when the available bandwidth drops. Unfortunately, several disadvantages to this approach exist. First, as every segment must start with an Instantaneous Decoder Refresh (IDR) frame to make it independent of other segments, a significant encoding overhead can be introduced [5]. Second, a larger number of HTTP GET requests are required to retrieve the segments, resulting in a larger network and server overhead. Moreover, at least one extra round-trip time (RTT) cycle is lost between subsequent requests, significantly reducing bandwidth utilization when the RTT is relatively high compared to the segment duration. This problem mainly arises in mobile networks, where the RTT can vary significantly, depending on the network carrier and the type of connection [6]. Furthermore, factors such as the type of WiFi network and the distance between client and server, have a possible impact on the RTT in the order of 100 ms [7]. As a consequence, it is typically infeasible to use segment durations below the one second range, and the resulting camera-to-display delay is in the order of tens of seconds. In this paper however, we introduce a novel technique to use super-short segments, using the push feature of the recently standardized HTTP/2 protocol.

Early 2012, the Internet Engineering Task Force (IETF) httpbis working group started the standardization of HTTP/2 to address a number of deficiencies in HTTP/1.1 [8, 9]. The new HTTP/2 standard was published in February 2015, and is now supported by major browsers such as Google Chrome, Firefox and Internet Explorer [10, 11]. 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 more important content and push content from server to client. In earlier work, we proposed a number of HTTP/2-based techniques that can improve the QoE in HAS, and particularly in live video streaming [12]. We mainly focused on a push-based approach, in which video segments are actively pushed from server to client in order to avoid idle RTT cycles. An overview of first results was provided for a fixed bandwidth scenario, showing that the approach is capable of significantly reducing the startup and end-to-end delay in high-RTT networks. In this paper however, we focus on the application of these techniques in mobile networks, where high variability in the available bandwidth is generally perceived. The adoption of our initially proposed approach is not straightforward in this case, because it assumes that pushed segments are delivered within a certain time interval. This assumption no longer holds when the available bandwidth is highly variable and sudden bandwidth drops occur. We therefore propose an improved technique which allows the deployment of a push-based approach in mobile networks, effectively using HTTP/2's newest features.

The contributions of this work are three-fold. First, we introduce a novel push-based approach, discussing its advantages for live video streaming in HAS. We show that in high-RTT networks, the startup and end-to-end delay can be significantly reduced, while a higher bandwidth utilization can be achieved. Second, we perform a detailed analysis on the encoding overhead for super-short segments, with relevant factors including the frame rate, the GOP length and the type of content. The trade-off between high responsiveness and a higher overhead is discussed, and a suitable segment duration is selected. Third, we present results from an extensive evaluation to characterize the gain of the proposed approach compared to state-of-the-art HAS over HTTP/1.1. Results are reported for a multi-client setup with variable bandwidth and latency, illustrating possible gains in mobile, high-RTT networks.

The remainder of this paper is structured as follows. Section 2 gives an overview of related work, focusing both on HAS and HTTP/2. The proposed push-based solution is presented in Sect. 3, followed by a discussion on the encoding overhead for super-short segments in Sect. 4. Detailed evaluation results on the push-based approach are provided in Sect. 5, before coming to final conclusions in Sect. 6.

2 Related Work

2.1 HTTP Adaptive Streaming

Techniques to improve the QoE perceived by the user for HAS services can be divided in client-based, server-based and network-based solutions [13]. A number of client-based solutions have recently been proposed in literature. Benno et al. propose a more robust rate adaptation heuristic for wireless live streaming [14]. By averaging the measured bandwidth over a sliding window, fluctuations are smoothed out, allowing the client to select a quality level that is sustainable and avoids oscillations. Petrangeli et al. propose a rate adaptation heuristic that strongly focuses on a new model for the QoE, taking into account the average video quality, its standard deviation and the impact of freezes in the decision taking process [15]. By expanding the heuristic to take into account a hierarchical fairness signal, fairness among clients can be induced without explicit communication between peers. Menkovski et al. propose the use of the SARSA(λ) technique at client-side to select the most appropriate quality level, based on the estimated bandwidth, the buffer filling and the position in the video stream [16]. Claeys et al. [17] propose the use of a Q-learning algorithm in the rate adaptation heuristic, based on the estimated bandwidth and the buffer filling. Results show that the client outperforms other deterministic algorithms in several network environments. Many other rate adaptation heuristics exist, but we refer to a survey by Seufert et al. for a more extensive view on the matter [13]. The above research attempts to increase the QoE through more intelligent client-side decisions, either increasing the average video quality, reducing the amount of switches or the occurrence of playout freezes for video on demand (VoD). In contrast, this research focuses on a reduction of the startup time and end-to-end delay for live streaming scenarios, taking into account the average video quality and the duration of playout freezes. Furthermore, the proposed approach is more general in nature: it can be extended to work on top of any existing rate adaptation heuristic.

Server-side solutions typically focus on new encoding schemes for HAS [13]. 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. This generally results in a storage overhead, increased bandwidth requirements and reduced caching efficiency. As suggested by Sánchez et al. [18], the adoption of scalable video coding (SVC) in HAS offers a solution to this problem. In SVC, redundancy is reduced because every quality level is constructed as an enhancement of the lower quality level. In the encoding process, lower quality 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. Although SVC provides an effective means to reduce content redundancy, it does introduce an encoding overhead of about 10% per layer. Since multiple requests are required to retrieve a single video segment, SVC-based solutions are even more susceptible to high RTTs. In addition, the commercial availability of SVC encoders and decoders often poses a problem. For these reasons, the application of SVC-encoded content in HAS is often questioned. In this paper, we focus on AVC-encoded content only.

Other server-side solutions exist as well. Akhshabi et al. [19] propose to use server-based traffic shaping to reduce video quality oscillations at the client. Their evaluations show a major reduction in terms of instability, practically stabilizing competing clients without a significant utilization loss. De Cicco et al. propose an approach based on feedback control, providing a Quality Adaptation Controller centralized at server side [20]. Based on a predefined target level, this controller attempts to keep the TCP send buffer at a bit rate close to the actual available bandwidth. Although the authors show that results are promising, the continuous bit rate control leads to a large, non-scalable workload at server side and does not take into account the client's actual buffer filling. In our proposed approach, quality decisions are located at the client, resulting in a distributed, scalable solution.

Network-based approaches often target IPTV solutions, where a dedicated network is used to provide the content to the user. Network elements are used to either optimize the QoE of single users, or to optimize the global QoE by providing fairness among clients. Bouten et al. avoid suboptimal distribution by introducing intelligence in the network that can override the client's local quality decisions [21]. In more recent work, the same authors use Differentiated Services (DiffServ) to give priority to the base layer segments in SVC [22]. As a result, the SVC-based client is more robust to video freezes, even when the total buffer size is decreased significantly to reduce the end-to-end delay for live video streaming. 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 [23]. 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. [17] in a multi-client scenario. More recently, the same authors propose priority-based delivery of HAS video [24]. In their approach, an OpenFlow controller is used to prioritize segments for clients which are close to buffer starvation. An extensive evaluation shows that their approach allows to reduce the total freeze time and frequency with up to 75%. Schierl et al. apply SVC-based streaming in a network environment with support for graceful degradation of the video quality when the network load increases [25]. The authors argue the need for Media Aware Network Elements (MANEs), capable of adjusting the SVC stream based on a set of policies specified by the network provider. More recently, MPEG proposed Server and Network Assisted DASH (SAND) [26]. In the suggested approach, a bi-directional messaging plane is used between the clients and other so-called DASH-Aware Network Elements (DANEs), in order to carry both operational and assistance information. In this way, the client can request the network to provide guaranteed bit rates for the video streaming session. Finally, Latré et al. propose an in-network rate adaptation heuristic, responsible for determining which SVC quality layers should be dropped, in combination with a Pre-Congestion Notification (PCN) based admission control mechanism [27]. In contrast to these works, this paper considers over-the-top video streaming only. As such, no extra middlebox functionality is required for the proposed approach.

2.2 The HTTP/2 Protocol

The new HTTP/2 standard was published as an IETF RFC in February 2015, mainly focusing on the reduction of latency in web delivery. The first draft for this protocol was based on SPDY, an open networking protocol developed primarily by Google [28]. Using this protocol, an average reduction of up to 64% was observed for the page load time. Other studies show that the mere replacement of HTTP by SPDY helps only marginally. Cardaci et al. [29] evaluate SPDY over high latency satellite channels. On average, SPDY only slightly outperforms HTTP. Erman et al. provide a detailed measurement study to understand the benefits of using SPDY over cellular networks [30]. They report that SPDY does not clearly outperform HTTP due to cross-layer dependencies between TCP and the cellular network technology. Similar results are reported by Elkhatib et al., who conclude that SPDY may both decrease or increase the page load time [31]. This is a consequence of the fact that SPDY's multiplexed connections last much longer than HTTP's, making SPDY more susceptible to packet loss. Similarly, Wang et al. show that SPDY can either reduce or increase the load time of web pages [32]. They conclude that next to a careful configuration of the TCP protocol, a reduction of the page load time may be achieved by changing the page load process using SPDY's server push. With respect to energy consumption, a recent study by Chowdhury et al. shows that HTTP/2 is more energy efficient than HTTP/1.1 when RTTs in the order of 30 ms or more are observed [33]. Since high RTTs are often observed in mobile networks, this is of significant importance for mobile devices with constrained battery life.

2.3 HTTP/2 for Multimedia Delivery

Mueller et al. are the first to evaluate the performance of DASH-based adaptive streaming over SPDY [34]. The existing HTTP/1.0 or HTTP/1.1 layer is replaced by SPDY, without any modifications to the HAS client or server. The authors show that if SPDY is used over SSL, the gains obtained by using header compression, a persistent connection and pipelining are almost completely cancelled out by the losses due the SSL and framing overhead. Wei et al. first explore how HTTP/2's features can be used to improve HAS [4]. By reducing the segment duration from 5 to 1 s, they manage to reduce the camera-to-display delay with about 10 s. An increased number of GET requests is avoided by pushing k segments after each request, using HTTP/2's server push. One disadvantage to this approach is 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. This results in an increased switching delay for the client and bandwidth overhead in the network. In later work, the authors show that the induced switching delay is about two segment durations and is independent of the value of k, while the introduced bandwidth overhead heavily depends on this value [35]. 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 [36]. Additionally, the authors propose a new approach for video adaptation, in which WebSocket over HTTP/2 is used to estimate the available bandwidth. In previous work, we proposed a number of techniques to improve the QoE in HAS, using on HTTP/2's stream prioritization, stream termination and server push [12]. Every technique has its own advantages, such as a higher bandwidth utilization and a gain of one or multiple RTT cycles in the client's startup phase. We thus proposed a *full-push* approach, in which several techniques are combined together. A first evaluation showed promising results, such as a lower startup time and end-to-end delay in high-RTT networks. In other work, we showed that the occurrence of playout freezes can be reduced through SVC-based streaming over HTTP/2, limiting the quality loss in high-RTT networks by pushing base layer segments to the client [37]. Although results are promising, the average quality is lower than for AVC-based HAS, and only a limited amount of quality layers can be used because of SVC's encoding overhead.

Except for our work on SVC, focus in the above research is mainly on reducing the live latency and the number of GET requests issued by the client. Results are shown for scenarios in which the available bandwidth is constant, or only changes every 20 s. Although significant improvements are obtained, it is not clear how the proposed approaches impact other aspects of the QoE, such as the average quality or the amount of playout freezes. Furthermore, there is no analysis of the encoding overhead introduced by using shorter video segments. In this paper, a new push-based approach is proposed to provide the server with more explicit feedback from the client, in order to avoid network congestion and buffer starvation when sudden bandwidth drops occur. The encoding overhead for shorter video segments is analyzed in detail, and evaluations are extended to a multi-client mobile network scenario with high variations in the available bandwidth and network latency.

3 Push-Based Approach

In this section, we first elaborate on the initially proposed push-based approach, in which video segments are actively pushed from server to client [12]. Its advantages and possible applications are discussed in detail, along with new challenges that arise in mobile, bandwidth-variable networks. We then propose an extension to the suggested approach, in which explicit feedback from the client is used to improve performance.

3.1 Full-Push Approach

In HAS, a video streaming session starts with the client sending a request for the video's media presentation description (MPD). This file contains information regarding the video segments, such as the duration, resolution and available bit rates. In live video, it also contains information regarding the timing of the video streaming session and the segments available on the server. Based on the contents of the MPD, the client then starts requesting video segments one by one, ramping up the buffer by downloading segments at the lowest quality. Once the buffer filling is sufficiently high-typically when a certain panic threshold is exceeded - further decisions regarding the video quality are made by the rate adaptation heuristic. The main drawback of this approach is that one RTT cycle is lost to download each segment, which has a significant impact on the startup time in high-RTT networks. An illustration of this behavior is presented in Fig. 2a, where the first phase of a live streaming session is shown. In the full-push approach, the server pushes m segments to the client as soon as the MPD request is received, where m corresponds to the

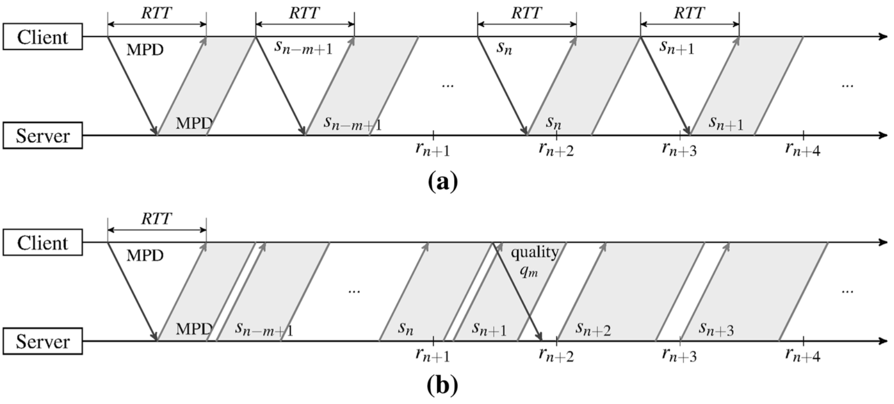


Fig. 2 An example live video scenario for HTTP/1.1 and HTTP/2, where the client requests m available segments to ramp up the buffer [12]. If the last released segment has index n, the first segment to play is thus $n-m+1$. Note that r_i denotes the release of segment i at server side, while s_i denotes its request/download by the client. Furthermore, *quality* q_j indicates that the server should change the quality of pushed segments to j. **a** Adaptive streaming using traditional HTTP/1.1. **b** Adaptive streaming using HTTP/2's server push

number of segments that fit into a preferred buffer size defined by the client. This of course requires that the server is aware of the relationship between the different HAS objects, and that the client can indicate its maximum buffer size in the MPD request. Since state-of-the-art rate adaptation heuristics ramp up the buffer by downloading segments at the lowest quality, it makes sense to push these at low 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 Fig. 2b, at least one RTT cycle is gained in the reception of the first video segment, and multiple RTT cycles are gained during the buffer rampup phase. Once the MPD and the first m segments are sent, the server either periodically pushes a new segment to the client at the specified quality level (VoD), or as soon as a new segment is available (live video). Every time a segment is completely received, the client estimates the perceived bandwidth and checks the buffer filling. Based on these parameters, the rate adaptation heuristic determines the most suitable video quality and if required, a request is sent to change the bit rate of pushed segments. Figure 3 shows a possible application of the full-push approach in a content delivery network (CDN) environment, both for VoD and for live video. The client controls the session by sending start, stop, pause, or resume messages under the form of an HTTP GET request to the server. In VoD, the client informs the server about the state of its buffer, indicating that segments must either be sent back-to-back or in a periodic way. In case of live streaming, the server ramps up the buffer by pushing the last m segments and from then on, pushes new segments to the client as soon as they are available.

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, video segments with a sub-second duration can be used, further reducing this delay. Using a smaller temporal buffer, the approach can effectively reduce the overall end-to-end delay as well. However, there are two disadvantages to a full-push approach for super-short segments.

First, while a segment n is being pushed from server to client, it is possible that new segments $n+1, \ldots, n+l$ are released. Since the rate adaptation heuristic of the client is only activated when a segment is completely received, these segments will

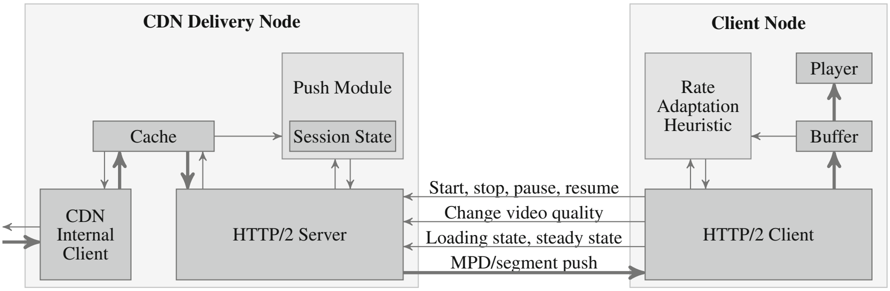


Fig. 3 Application of the full-push approach in a CDN environment. The client controls the session by sending request messages under the form of HTTP GET requests to the server

all be pushed to the client at the same video quality as segment *n*. This entails that, when sudden bandwidth drops occur, the server will not immediately lower the quality level to match the available bandwidth within the network. This will generally result in more packet queuing in the network buffers, leading to a higher risk of buffer starvation and thus to a lower QoE. To overcome this issue, Sect. 3.2 covers a way to limit the maximum number of segments in flight. In this way, network congestion is avoided, as the server can proactively adjust the pushed video quality to the available bandwidth.

Second, since every segment has to start with an IDR frame, a higher bit rate is required to achieve the same video quality. As such, an encoding overhead is required to achieve the same visual quality for content provided with a smaller Group of Pictures (GOP). In Sect. 4, we investigate the encoding overhead for eight different videos and select an appropriate minimal segment duration to evaluate the proposed push-based approach.

3.2 Full-Push Approach with Acknowledgments

In the full-push approach, the server can theoretically push an indefinite number of (high-quality) segments to the client. When the available bandwidth suddenly drops, a large number of packets will thus be queuing in the network. However, the intention of the HTTP/2-based approach is merely to push segments in order to bridge the idle RTT cycles that occur when segments are pulled by the client. To achieve this, the server does not need to send an indefinite number of segments; it is sufficient to push a maximum of *k* segments consecutively, where *k* depends on the network's RTT and the segment duration of the video. For this reason, we propose an improved approach in which the maximum number of segments in flight is limited to a certain value *k*. Note that this is no standalone solution; it represents a possible, complimentary dimension of the initially proposed push-based approach.

The suggested approach is illustrated in Fig. 4, for a value of $k = 2$. Similar to the full-push approach, the client first sends a GET request to fetch the MPD, indicating its preferred buffer size. The server pushes the first *m* segments at the lowest quality, where *m* corresponds to the number of segments that fit in the buffer size defined by the client. Then, segments are periodically pushed as soon as they become available. The main difference is, however, that the server stops pushing

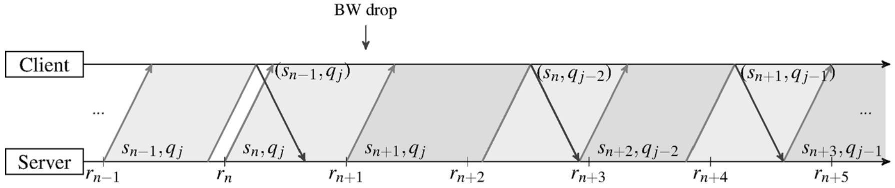


Fig. 4 Illustration of HTTP/2's server push with explicit acknowledgments. A value of $k = 2$ is used in this scenario, so that either 0, 1 or 2 segments are in flight (indicated in *white*, *light gray* and *dark gray* respectively)

new segments until the reception of the *k*th last segment is acknowledged by the client. To this end, a sliding window is used on the server's application layer, defining which segments can and cannot yet be pushed. At client side, the rate adaptation heuristic determines its preferred quality level for future segments once a new segment has completely arrived. This quality level and the received segment's index are then passed along to the server, which moves the sliding window one segment to the right and pushes the next segment as soon as it is available (if not already). With respect to the parameter k, the following rules of thumb apply:

1. For low RTTs and high segment durations, high bandwidth utilization is achieved even without the presence of a push-based approach. Low values of k should be used in order to minimize network congestion;
2. For high RTTs and low segment durations, a large value of k should be applied in order to maximize bandwidth utilization. Unfortunately, this is expected to have a negative impact on the freeze duration and frequency as well.

The value of k should thus be finetuned based on the state of the network and the video's segment duration. In Sect. 5, we will show that the following rule of thumb can be used to calculate the optimal value for k, given the RTT and the segment duration seg:

$$k = \begin{cases} ceil\left(\frac{RTT}{seg}\right) + 1, & \text{if } \frac{RTT}{seg} > a \\ 1, & \text{otherwise.} \end{cases} \qquad (1)$$

In this equation, the value of k is directly proportional to the ratio of the RTT and the segment duration, unless this ratio is lower than a threshold value a. The reasoning behind this is as follows. When the ratio of the RTT and the segment duration is small, little improvement of the average video quality is expected when a push-based approach is used. Since pushing multiple segments increases the chances of buffer starvation when a sudden bandwidth drop occurs, it is more appropriate to only pull the segments in this case. Using $k = 1$, one segment is pushed for every request/acknowledgment, which indeed corresponds to a pull-based approach. If the ratio of the RTT and the segment duration is large, however, a significant gain can be achieved for the average video quality. In this case, idle RTT cycles should be avoided by actively pushing new video segments. In Sect. 5, we will show that $a = 0.2$ is an appropriate value for the ratio's threshold.

4 Impact of the Segment Duration on the Encoding Overhead

Since our proposal is to use video segments with a sub-second duration, it is important to analyze the induced encoding overhead. To perform this analysis, eight different videos are considered: Big Buck Bunny[1] (BBB), Earth from Space[2] (EFS),

[1] https://peach.blender.org/.

[2] https://www.youtube.com/watch?v=n4IhCSMkADc.

Netflix' El Fuente[3] (NEF), Sintel[4] (STL), Tears of Steel[5] (ToS), Elephant's Dream[6] (ED), a benchmark performing testing video on Forza Motorsport 6 Apex[7] (FZA) and a 10-minute part of a soccer game from the 2014 World Championship[8] (SOC). All content is available in Full HD (1920 × 1080 px), and comes with a minimal frame rate of 24 FPS. In our analysis, segment durations from 125 ms up to 8 s are considered, using the FFmpeg library[9] to segment the video. To allow each segment to be decoded independently, every segment starts with an IDR frame and the Group Of Pictures (GOP) length is set accordingly. To assess the impact of shorter GOP lengths on the compression performance, the encodings for different segment durations are set to target the same visual quality and allow a subsequent overhead in the achieved nominal bit rate. To realize this, the Constant Rate Factor (CRF) rate control implemented in the x264 encoder[10] is used. The obtained encodings for the same nominal rates but different segment durations, have the same visual quality, measured in terms of Peak-Signal-to-Noise-Ratio (PSNR), with deviations smaller than 0.205 dB for all content bar BBB (1.045 dB).

Figure 5 shows the encoding bit rate relative to a segment duration of 2 s, for the BBB, EFS and NEF content provided in Full HD with frame rates of 24, 30 and 60 FPS. The encoding overhead is significantly lower for higher frame rates, which is a consequence of the higher GOP length: for a segment duration of 500 ms, a GOP length of 12, 15 and 30 is used for 24, 30 and 60 FPS respectively, resulting in relatively less IDR frames. The obtained results are also strongly content-dependent: an overhead of 70.8% is obtained for NEF at 24 FPS, while an overhead of 195.9% is obtained for BBB. This is because the former is a video showing actually captured footage with high-detail shots, whereas the latter is synthetically produced 3D video.

Figure 6 shows the relative encoding bit rate for all considered videos, provided in Full HD at 24 FPS. For all content bar BBB, the encoding overhead is between 62.0 and 106.1% for a segment duration of 125 ms, between 28.7 and 49.9% for 250 ms, between 12.1 and 20.4% 500 ms and between 3.9 and 6.6% for 1 s. Based on these values, we conclude that a minimal segment duration of 500 ms should be used in order to limit the encoding overhead and achieve a similar video quality when the available bandwidth is constrained. Evaluation results in the next section will confirm this conclusion, yet also show that lower segment durations can result in an even lower video startup time and end-to-end delay. Since our focus is on live streaming scenarios, the soccer video will be used to evaluate results for the push-based approach. For this video, average overhead values of 87.4, 41.9, 18.0 and 5.8% are obtained for a segment duration of 125, 250, 500 ms and 1 s respectively.

[3] https://www.netflix.com/title/70297450.

[4] https://durian.blender.org/.

[5] https://mango.blender.org/.

[6] https://orange.blender.org/.

[7] https://www.youtube.com/watch?v=d-Jgf6rtEg8.

[8] https://www.youtube.com/watch?v=qOHd20F0e9k.

[9] https://ffmpeg.org.

[10] http://www.videolan.org/developers/x264.html.

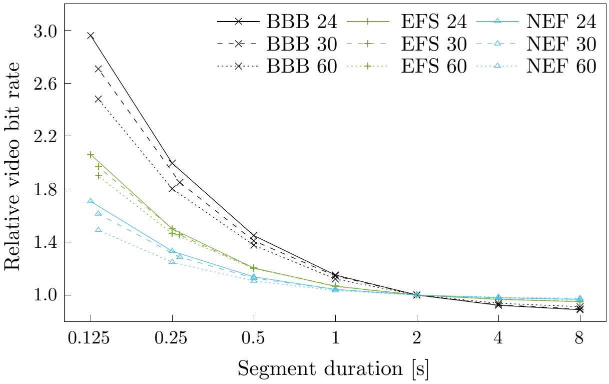


Fig. 5 Encoding bit rate relative to a segment duration of 2 s as a function of the segment duration, for frame rates of 24, 30 and 60 FPS

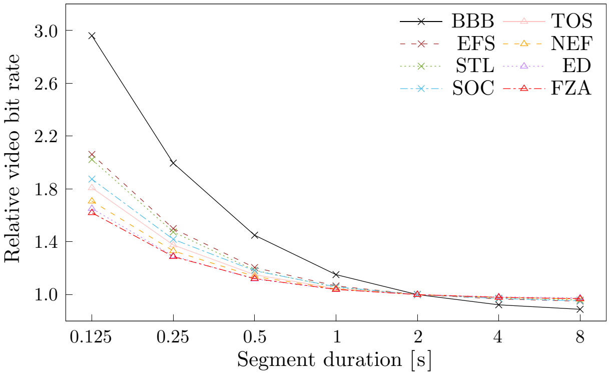


Fig. 6 Encoding bit rate relative to a segment duration of 2 s as a function of the segment duration, for a frame rate of 24 FPS

A frame rate of 24 FPS is used to show the minimal gains of the approach; even better results can be achieved with a frame rate of 30 or 60 FPS, currently supported by live stream providers such as YouTube.[11]

5 Evaluation and Discussion

To illustrate the possible gains of the push-based approach presented in Sect. 3, results are evaluated for different network conditions and video segment durations. The experimental setup is presented below, followed by a discussion on the evaluated rate adaptation heuristics and a detailed overview of the obtained results.

[11] https://www.youtube.com.

5.1 Experimental Setup

The considered video in our evaluation is a part of a soccer game,[12] which has a total length of 596 s and comes with a frame rate of 24 FPS. Similar as in the previous section, the video is segmented using segment durations ranging from 125 ms to 8 s, corresponding to GOP lengths ranging from 3 to 192. The video is encoded at six quality levels, with resolutions of $256 \times 144, 428 \times 240, 640 \times 360, 856 \times 480, 1280 \times 720$ and 1920×1080 px, using x264 with a CRF of 23. Note that there's a lot of variability among bit rates for the different segments, as illustrated in Fig. 7 for a segment duration of 2 s.

To stream the content, the network topology in Fig. 8 is emulated using the MiniNet framework.[13] It consists of 30 clients, streaming video from a dedicated HAS server. To evaluate the proposed approaches in a realistic scenario, bandwidth and latency patterns are applied for every client. Patterns for the available bandwidth are extracted from an open-source dataset, collected by Riiser et al. on a real 3G/HSDPA network [38]. The average available bandwidth in the 30 traces is 2358 kb/s, with a standard deviation of 1387 kb/s. The minimum available bandwidth is set to 300 kb/s, in order to provide the minimal video streaming service. An example trace is shown in Fig. 9a, illustrating the high bandwidth variability over time. As for latency, first evaluations are performed with a statically defined value ranging between 0 and 300 ms in order to assess its impact on the performance of a number of rate adaptation heuristics and the optimality of the number of segments in flight k. In a final evaluation, the latency is dynamically changed according to measurements performed in a real 3G/HSPA network, illustrated in Fig. 9b. To collect a trace for the perceived latency, we hosted a dedicated server in iLab.t's Virtual Wall infrastructure.[14] Sending a ping request every second from a smartphone connected over 3G/HSPA, the perceived latency was measured and logged for a total of 17 min. For each client, the trace is looped and latency values are set according to a random starting point. The average latency is 232.2 ms, with a standard deviation of 12.5 ms.

Values for variable bandwidth and latency are introduced through traffic shaping with *tc* on the client-side switches. The bandwidth on links L_C is set to 10 Mb/s, while the bandwidth on the link L_S is set to $30 \times L_C = 300$ Mb/s. Since the maximum bandwidth in the traces is 6.2 Mb/s, the bottleneck thus resides with the clients and the actual bandwidth corresponds to the bandwidth perceived at the time of trace collection.

5.1.1 Server-Side Implementation

The HAS server implementation is based on the Jetty web server,[15] which was recently expanded to provide support for HTTP/2. Jetty's HTTP/2 component

[12] Available at http://users.ugent.be/~jvdrhoof/content.

[13] http://mininet.org/.

[14] http://ilabt.iminds.be/iminds-virtualwall-overview.

[15] https://webtide.com/.

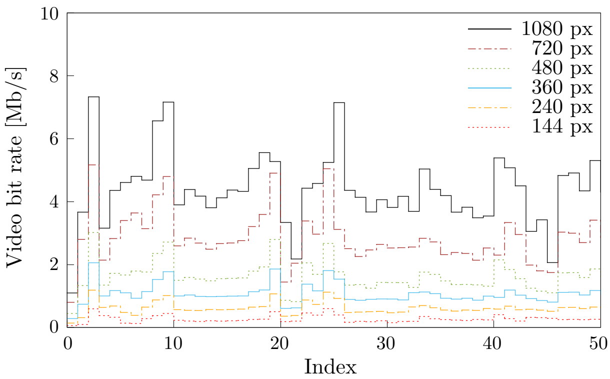


Fig. 7 Video encoding bit rates for the first 50 segments of the Soccer video, for a segment duration of 2 s

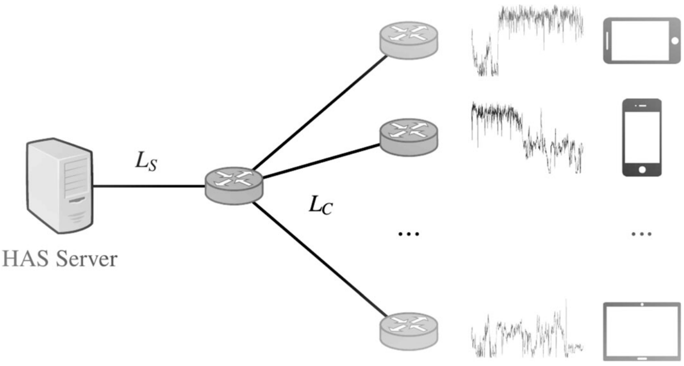


Fig. 8 Experimental mininet setup

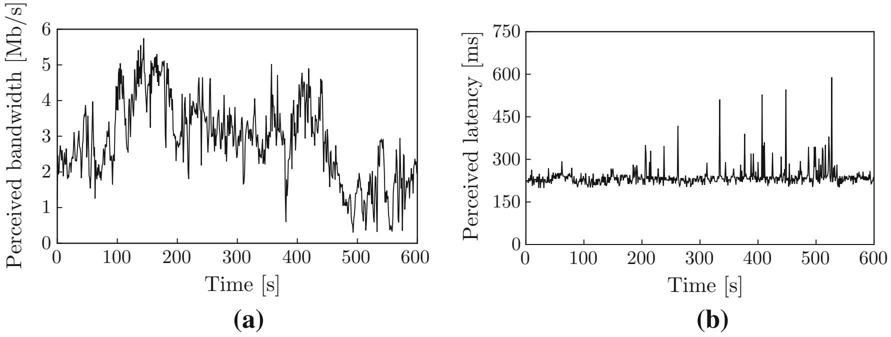


Fig. 9 Perceived bandwidth in one of the 3G/HSDPA traces collected by Riiser et al. [38], along with values for the perceived latency collected in a 3G/HSPA network. **a** Perceived bandwidth. **b** Perceived latency

allows to define 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, where the required JavaScript and CSS files, images and other content can immediately be pushed. However, since we target a live stream scenario, not all segments are available when the initial 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 temporal 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, where m corresponds to the number of segments that fit in the indicated buffer. 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 the segment is pushed to the corresponding client. When the client wants to change the quality level at which the segments are pushed, a new request is issued and the quality level is updated at server-side accordingly.

5.1.2 Client-Side Implementation

The HAS client is implemented on top of the libdash library,[16] 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 are made. First, an HTTP/2-based connection is added to enable the reception of pushed segments. The nghttp2 library[17] is used to set up an HTTP/2 connection over SSL. Second, the rate adaptation heuristic is modified to recalculate the quality level every time the push stream of a pushed segment is closed. While a GET request is required for every segment in HTTP/1.1, 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. In HTTP/1.1, this estimation is based on the total download time, which naturally includes the time to send the GET request. Note that by default, the available bandwidth is estimated by the Exponentially Weighted Moving Average (EWMA) over the observed download rates. It is worth noting that major browsers such as Google Chrome, Moxilla Firefox and Internet Explorer all provide support for HTTP/2 [11]. Although a standalone client is used in this paper, a browser-based DASH.js reference player has been released by libdash as well.[18] Provided that some changes to the implementation are made, the proposed approach can thus be applied in any browser with support for HTTP/2.

[16] https://github.com/bitmovin/libdash/.

[17] https://nghttp2.org/.

[18] http://dashif.org/reference/players/javascript/1.4.0/samples/dash-if-reference-player/.

5.2 Rate Adaptation Heuristics

To achieve a baseline for adaptive streaming over HTTP/1.1, a number of state-of-the-art rate adaptation heuristics were embedded at client side. Evaluated heuristics are the MSS heuristic developed by Microsoft [39], the FESTIVE heuristic developed by Jiang et al. [40] and the FINEAS heuristic[19] developed by Petrangeli et al. [15].

In the MSS heuristic, the next quality level is selected based on the buffer filling and the perceived bandwidth. The most important parameters are the buffer size and the panic, lower and upper thresholds, which actively steer the buffer filling towards a value between the lower and upper threshold. A lower quality level is selected when the buffer filling drops below the lower threshold, while a higher quality level is selected when the buffer filling exceeds the upper threshold. When the buffer filling is lower than the panic threshold, the rate adaptation heuristic immediately selects the lowest quality level, in an attempt to avoid buffer starvation. In accordance with conclusions drawn by Famaey et al. [41], panic, lower and upper thresholds of respectively 25, 40 and 80% of the total buffer size were selected.

In the FESTIVE heuristic, the first 20 segments are downloaded at the lowest quality, because initially there is not enough information on the available bandwidth [40]. From segment 21 on, the rate adaptation heuristic computes a reference quality level q_{ref} based on the available bandwidth (defined as the harmonic mean of the download rates for the last 20 segments) and the last selected quality level q_{cur}. A gradual switching strategy is applied, so that the reference quality can only switch to the next lower or higher quality layer. Furthermore, a switch from quality level m to $m + 1$ can only occur once at least m segments have been downloaded at quality m. Given q_{cur} and q_{ref}, the heuristic then calculates a certain metric score for the session’s stability and efficiency, and finally selects the most appropriate quality level.

In the FINEAS heuristic, the goal is to maximize the user’s QoE [15]. This is achieved by intelligently selecting the next quality level, pursuing a high average quality level while limiting the number of quality switches and avoiding play-out freezes. Quality selection is based on a utility function, designed to be a metric for the QoE. Used parameter values for this evaluation are a panic threshold of two segments, a targeted buffer filling of 80% and a quality window of 70 s, selected after tweaking the heuristic for a buffer size of 10 s. Note that, in order to limit the number of switches for super-short segments, upward switching is only allowed every 2 s.

5.3 Evaluation Metrics

The following evaluation metrics are considered. First, the average video quality, expressed as the average quality level (1–7) perceived during a streaming session. Second, buffer starvation, defined by the frequency and total duration of playout freezes. Third, the initial startup delay, which is defined as the time between

[19] The in-network computation performed by the authors has not been implemented in this work.

requesting the live video at client-side and the playout of the first video segment. Fourth, the server-to-display delay, defined as the time between the release of a segment at server-side and its playout at client-side, plus the segment duration. This is not the same as the camera-to-display delay, as the content is not captured in real-time. Note that segments are not skipped during playout, so every freeze results in a larger buffer and total server-to-display delay. When no freezes occur, the initial and final server-to-display delay are the same.

To evaluate the impact of the proposed approach on these metrics, the considered video is streamed by 30 different clients; results are therefore shown using the observed averages and the corresponding 95% confidence intervals.

5.4 Evaluation Results

In this Section, we first show results for HTTP/1.1, highlighting the advantages that the proposed push-based approach can bring. Second, a parameter analysis is performed to find the optimal value for the parameter *k*, based on the segment duration and the network's RTT. Third, the impact of the buffer size on the QoE is evaluated, showing results for the average video quality, freeze time, startup time and server-to-display delay. Finally, a detailed comparison of results is made between the proposed HTTP/2-based approach and traditional HAS over HTTP/1.1.

5.4.1 HAS over HTTP/1.1

With respect to the considered rate adaptation heuristics, Fig. 10 shows the average video quality and freeze time for MSS, FESTIVE and FINEAS as a function of the latency. The MSS heuristic decreases the video quality in a conservative way, only allowing a decrease of one level at a time as long as the panic threshold is not exceeded. Because of this, reaction to changes in the available bandwidth are generally slow, resulting in a relatively large average freeze time. In the FESTIVE heuristic, the video quality is decreased one level at a time as well.

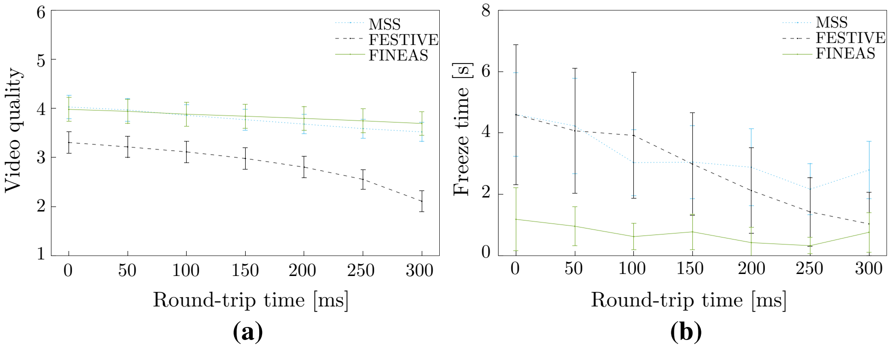


Fig. 10 Average video quality and freeze time for the MSS, FESTIVE and FINEAS heuristics, as a function of the RTT for a segment duration of 2 s and a buffer size of 10 s. **a** Video quality. **b** Freeze time

Furthermore, the estimated bandwidth is based on the harmonic mean over the last 20 samples, instead of the more aggressive EWMA. As such, the average freeze time is similar to the one for MSS. The average video quality is however significantly lower, because even with sufficient bandwidth it takes a long time for the heuristic to reach the highest quality: 20 segments are first requested at the lowest quality, another 20 are requested at the intermediary levels (2–6). The FINEAS heuristic allows to change the quality level with more than one level at a time, resulting in a more aggressive reaction to changes in the available bandwidth. Furthermore, buffer starvation is actively prevented by taking into account both the current buffer filling and the estimated download times of future segments. Comparing results with MSS, a similar average video quality is observed, yet with a significantly lower average freeze time. FINEAS thus leads to the best results, and will be used to evaluate the proposed HTTP/2-based approach in the following sections.

Figure 11 shows results for the FINEAS heuristic over HTTP/1.1, for an increasing RTT and for all considered segment durations. From Fig. 11a, it is clear that super-short segments suffer significantly from large RTTs: for a segment

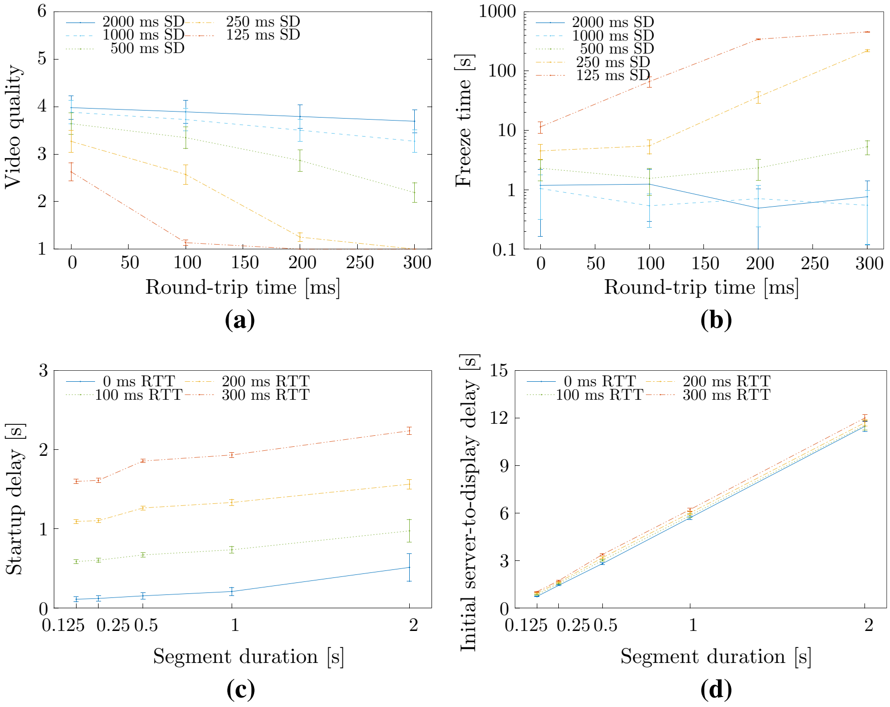


Fig. 11 Impact of the RTT on the average video quality and freeze time for the FINEAS heuristic (*top*), and the impact of the segment duration (SD) on the startup delay and server-to-display delay (*bottom*). While super-short segments can result in a lower startup and end-to-end delay, they result in a low video quality and high freeze time for high RTTs. **a** Video quality. **b** Freeze time. **c** Startup delay. **d** Initial server-to-display delay

duration of 125 ms, the lowest quality is always selected for RTTs above 200 ms. For a segment duration of 2 s however, the average video quality is only reduced with 7.2% for an RTT of 300 ms. Looking at the total freeze time in Fig. 11b, a significant increase is observed for a segment duration of 125 ms and RTTs above 200 ms: in this case, a freeze inevitably occurs for every downloaded video segment. Furthermore, since the buffer size is five times the segment duration (and thus only 625 ms for a segment duration of 125 ms), the freeze time for super-short segments is relatively high, even when a negligible RTT is considered. From these results, we conclude that a segment duration of 2 s should be used in order to achieve an appropriate video quality and freeze time in mobile, high-RTT networks. Note that similar results are achieved for the MSS and FESTIVE heuristics, omitted here due to space constraints.

Despite these disadvantages, there are clear advantages as well. For one, Fig. 11c shows that the startup delay increases significantly for larger segment durations. This is because more data needs to be put on the wire for the first video segment, and because the underlying TCP slow start requires multiple RTT cycles to transfer the required files from server to client. More importantly, Fig. 11d shows that the initial server-to-display delay increases linearly with the buffer size. Since both the startup and the end-to-end delay are important factors in live video streaming, our goal is to realize the advantages of super-short segments, while still providing the content at an acceptable video quality. The following sections will therefore focus on the applicability of super-short segments, using the proposed push-based approach to avoid idle RTT cyles.

5.4.2 Impact of the Parameter k

In Sect. 3.2, Eq. (1) was proposed as a rule of thumb for the maximum number of segments in flight k. To test the validity of this equation, all combinations of the parameter configurations in Table 1 were evaluated, applying latency to the links L_C in the experimental setup. Figure 12a, c show the impact of the value of k on the average quality, for a segment duration of 125 and 500 ms respectively. In the former case, a total of two, three and four segments must be pushed for an RTT of 100, 200 and 300 ms respectively, in order to achieve a video quality similar as for a negligible RTT. In the latter case, however, it is sufficient to push a maximum of two segments at the same time. With respect to the average freeze time, shown in Fig. 12b, d, two observations can be made. First, for super-short segments, the average freeze time is significantly reduced when multiple segments are pushed.

Table 1 Evaluated configurations for the parameter k

Parameter	Evaluated values
Segment duration (s)	0.125, 0.25, 0.5, 1, 2
Latency (ms)	0, 100, 200, 300
k	1, 2, 3, 4, 5, ∞

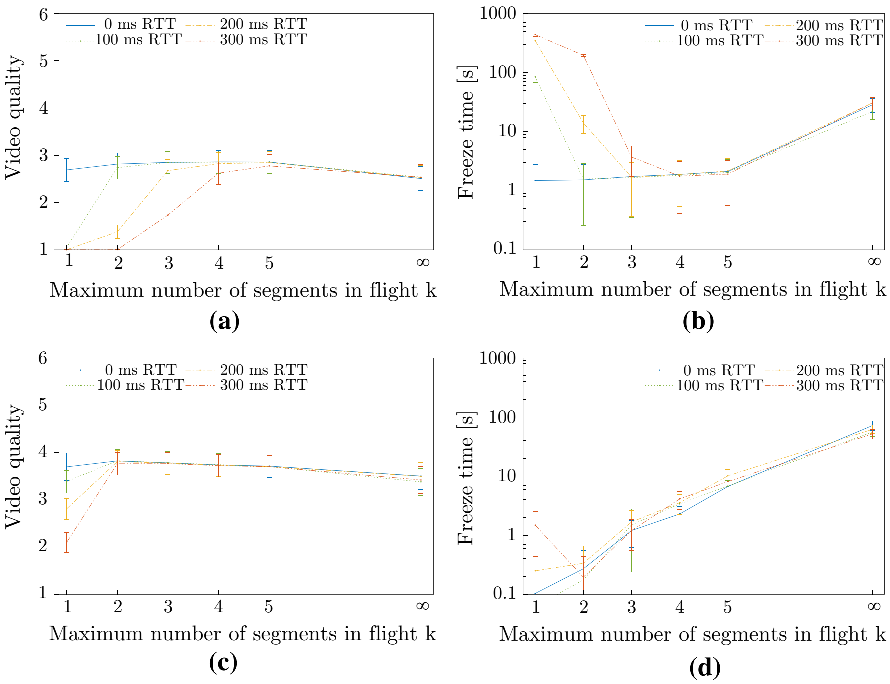
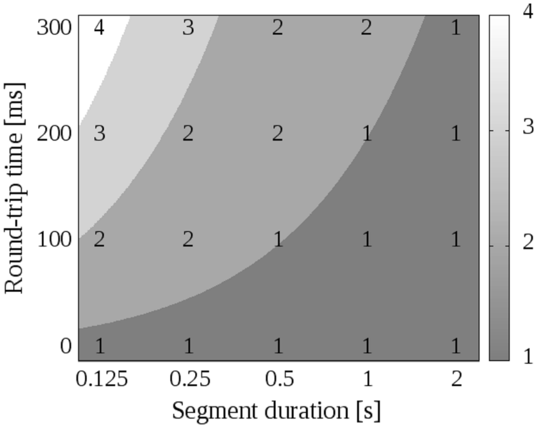


Fig. 12 Impact of the RTT and impact of the parameter *k* on the average video quality and freeze time, for the FINEAS heuristic with a segment duration of 125 ms (*top*) and 500 ms (*bottom*). The optimal value for *k* depends both on the segment duration and the network's RTT, with higher values for lower segment durations and high RTTs. **a** Video quality, SD 125 ms. **b** Freeze time, SD 125 ms. **c** Video quality, SD 500 ms. **d** Freeze time, SD 500 ms

This is because bandwidth is used more efficiently, so that no freeze occurs for every single segment. For a segment duration of 500 ms, however, a reduction is less apparent. Second, once a certain threshold for k is exceeded, the average freeze time increases significantly. This is because more segments are allowed to be in flight, resulting in a slower quality reduction when sudden bandwidth drops occur. This is in line with the reasoning in Sect. 3.2, where the maximum number of segments in flight is based on the segment duration and the network's RTT, and shows that the initial approach ($k = \infty$) is not suitable in mobile, highly variable networks.

For all combinations of the evaluated segment durations and RTTs in Table 1, the lowest value for k was selected that resulted in a comparable video quality compared to a scenario with negligible RTT. Figure 13 shows the obtained values for these configurations, along with the output of Eq. (1). A nearly perfect match is obtained for $a = 0.2$, showing the suitability of the proposed rule of thumb. In the remainder of this paper, a value of $k = 2$ will be used to push video segments with a segment duration of 500 ms.

Fig. 13 Optimal values for the parameter *k* in the configurations of Table 1, along with the output of Eq. (1) for a value of $a = 0.2$ in colors from *dark grey*($k = 1$) to *white*($k = 4$)



5.4.3 Impact of the Buffer Size

Having analyzed different heuristics and optimal values for the parameter *k* as a function of the latency, we illustrate the gains of the proposed approach below. In this experiment, we analyze the impact of the buffer size on the different evaluation metrics, and show that the proposed approach allows to reduce the buffer size and the end-to-end delay significantly. To this end, temporal buffer sizes ranging from 2 to 10 s are used. A segment duration of 2 s is used for HTTP/1.1, while a segment duration of 500 ms is used for HTTP/2 ($k = 2$). In the former case, panic thresholds and targeted buffer filling are selected based on observations by Petrangeli et al. [15], with values presented in Table 2. In the latter case, multiples of 500 ms are selected for the panic threshold and buffer target respectively. Note again that the maximum buffer size can change over time: no segments are skipped when playout freezes occur, thus increasing the end-to-end delay and the amount of content the server can push to the client.

Figure 14a shows the average video quality in terms of the initial buffer size. For HTTP/1.1, the lowest quality is most frequently selected for a buffer size of 2 s, since the panic threshold is almost always exceeded. For higher buffer sizes, the average quality gradually increases. For HTTP/2, however, the average video

Table 2 Evaluated panic thresholds (PT) and targeted buffer filling (BF) for the FINEAS heuristic, both for HTTP/1.1 (1) and HTTP/2 (2) with different values of the buffer size

Buffer size (s)	PT-1 (s)	BF-1 (s)	PT-2 (s)	BF-2 (s)
2	2	2	1	1.5
4	2	2	2	3
6	4	4	2.5	4.5
8	4	6	3.5	6
10	4	8	4	8

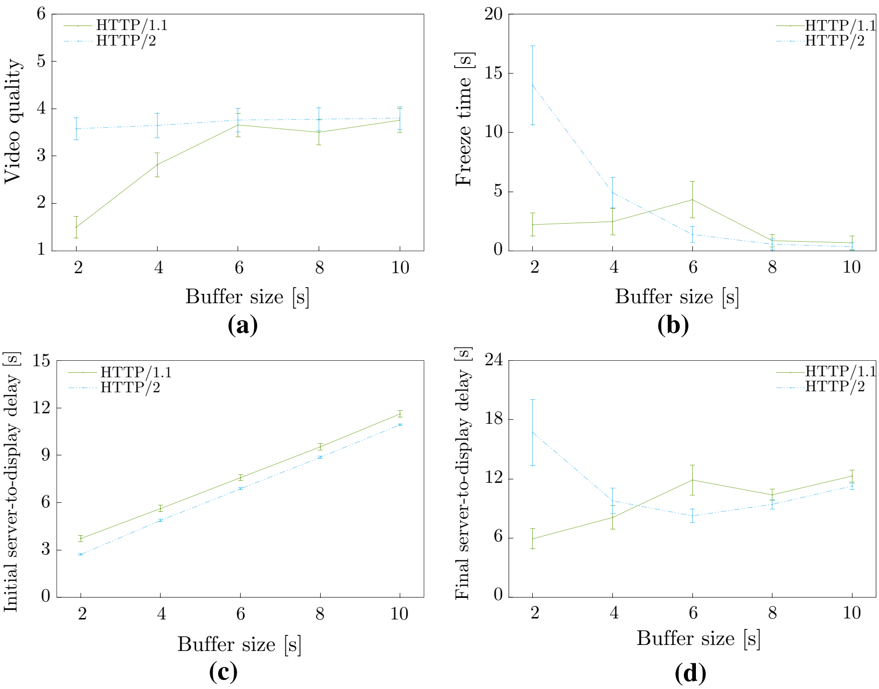


Fig. 14 Impact of the buffer size on the video quality, the freeze time and the server-to-display delay for the FINEAS heuristic, for an RTT of 300 ms. A segment duration of 2 s is used for HTTP/1.1, while a segment duration of 500 ms is used for HTTP/2 ($k = 2$). **a** Video quality. **b** Freeze time. **c** Initial server-to-display delay. **d** Final server-to-display delay

quality is almost unaffected by the buffer size. Figure 14b shows the average freeze time per episode, indicating a decreasing trend for larger buffer sizes. When a buffer size of 2 or 4 s is used, a large number of playout freezes occur for HTTP/2. This is because the rate adaptation heuristic can download the video at high quality (the panic threshold is often not exceeded), yet a buffer this small cannot cope with the high variability of the perceived bandwidth and the video segment's size. This is less of a problem for HTTP/1.1, simply because the heuristic most often downloads the video at the lowest video quality. Once the buffer threshold exceeds 6 s, higher quality representations can be downloaded both for HTTP/1.1 and HTTP/2. Since the latter allows to react faster to changes in the available bandwidth, a lower playout freeze time is observed. The initial server-to-display delay is shown in Fig. 14c, increasing linearly with the selected buffer size. Figure 14d, on the other hand, shows the server-to-display delay at the end of the video streaming session. Since no segments are skipped in our setup, every playout freeze results in an increase of the end-to-end delay. As such, the total delay corresponds to the sum of the initial delay and the episode's total freeze time. Even though a buffer size in the order of a few seconds can be used to reduce the initial delay, it cannot cope with the highly variable bandwidth and segment sizes. As a result, the total delay is significantly higher than for a buffer size of 6 s.

From these results, we conclude that a minimal buffer size of 10 s is required for HTTP/1.1, in order to obtain an acceptable video quality with a relatively low amount of playout freezes. Because of the shorter segment duration, a buffer size of 6 s can be used for HTTP/2, reducing the end-to-end delay while limiting the total freeze duration.

5.4.4 Discussion

Table 3 summarizes results for traditional HAS over HTTP/1.1 and our proposed push-based approach. The average quality for MSS and FESTIVE is significantly lower than for the FINEAS heuristic (−3.9 and −32.4% respectively), while the average quality for the proposed push-based approach is more or less the same. Furthermore, while MSS results in an average freeze time close to 3 s, the FINEAS heuristics results in a freeze time well below 2 s both for HTTP/1.1 and HTTP/2. With respect to the video quality and freeze time, we thus conclude that both FINEAS approaches perform well. For the startup delay, however, a reduction of 0.57 s (−31.2%) is obtained. This is because the first segment is immediately pushed along with the MPD, and because TCP slow startup requires fewer RTT cycles to transfer the first, super-short segment. As for the server-to-display delay, a reduction of 4.04 s (−32.9%) is obtained, which is beneficial in a live streaming scenario. Given that significantly better results are obtained for the startup and server-to-display delay, and similar results are obtained for the video quality and freeze time, we conclude that the proposed push-based approach with super-short segments can significantly improve the QoE for live streaming in this scenario.

The reported results specifically target a mobile 3G/HSPA network, with relatively high values for the perceived latency. Compared to these networks, 4G/LTE networks allow to significantly reduce latency. For Belgian providers such as Base, Proximus and Mobistar, OpenSignal reports a reduction from 214 to 94 ms, from 241 to 64 ms and from 229 to 77 ms respectively [6]. Although the proposed approach should still result in lower startup times and end-to-end delay, the gains will be significantly lower. Highly congested networks, in contrast, introduce additional latency through network queueing. This can have a significant impact on the overall perceived latency, and depending on the variability of the network load, result in packet delay variation. In such a scenario, the parameter k could be changed dynamically, similar to the approach proposed by Wei et al. [35].

Table 3 Performance summary for the different heuristics and the proposed push-based approach

Approach	Heuristic	Video quality	Freeze time (s)	Startup time (s)	Server-to-display delay (s)
HTTP/1.1	MSS	3.61 ± 0.20	2.68 ± 1.00	1.82 ± 0.13	14.31 ± 1.04
HTTP/1.1	FESTIVE	2.54 ± 0.21	1.54 ± 1.09	1.82 ± 0.12	13.32 ± 1.14
HTTP/1.1	FINEAS	3.76 ± 0.25	0.68 ± 0.57	1.82 ± 0.13	12.30 ± 0.60
HTTP/2	FINEAS	3.76 ± 0.25	1.38 ± 0.67	1.25 ± 0.04	8.26 ± 0.69

Average values are reported, along with the 95% confidence intervals. Note that a segment duration of 2 s and a buffer size of 10 s are used for HTTP/1.1, while a segment duration of 500 ms and a buffer size of 6 s are used for HTTP/2

It should be noted that results are presented for H.264-encoded video with a frame rate of 24 FPS. As shown in Sect. 4, using higher frame rates results in a relatively lower encoding overhead, and thus in a higher relative video quality. This leads to a higher encoding bit rate, which can prove unsuitable under certain network conditions. An alternative can also be provided through H.265, which in some cases allows to reduce the encoding bit rate by half compared to H.264 [42]. Numerous other possibilities exist within adaptive streaming, but cannot all be considered in this evaluation.

6 Conclusions and Future Work

In this paper, focus is on improving the user's Quality of Experience (QoE) in HTTP Adaptive Streaming (HAS) in mobile, high round-trip time (RTT) networks. To this end, we proposed a novel push-based approach for HAS over the recently standardized HTTP/2 protocol. Using this approach, available video segments are actively pushed from server to client, effectively eliminating idle RTT cycles. This enables the use of segments with a sub-second duration, referred to as *super-short* segments, which not only allow the client to start the video playout faster, but also allow to significantly reduce the buffer size and thus the end-to-end delay. In contrast with previous work, we proposed a means to effectively limit the amount of segments on the fly, and performed an analysis of its optimal value for different segment durations and network latencies. Furthermore, we discussed in detail the encoding overhead introduced by shorter segments, and evaluated the proposed approach using highly variable bandwidth and latency traces collected in real 3G networks. Using a segment duration of 500 ms and a buffer of a mere 6 s, reductions of 31.2 and 32.9% are obtained for the startup time and the server-to-display delay respectively, compared to traditional HAS over HTTP/1.1 with a segment duration of 2 s and a buffer size of 10 s.

Future work includes a real-life study of the overall approach using subjective measurements, such as the Mean Opinion Score, which could reveal interesting insights. An interesting future research path includes maximizing the average QoE and fairness for competing HAS clients in congested networks, using a dynamic approach to change the parameter k based on changing network conditions.

Acknowledgements 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, and within 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. https://www.sandvine.com/downloads/general/global-internet-phenomena/2016/global-internet-phenomena-report-latin-america-and-north-america (2016). Accessed 21 Feb 2017

2. Mok, R., Chan, E., Chang, R.: Measuring the quality of experience of HTTP video streaming. In: IFIP/IEEE International Symposium on Integrated Network Management, pp. 485–492 (2011)
3. Stockhammer, T.: Dynamic adaptive streaming over HTTP: standards and design principles. In: Proceedings of the 2nd Annual ACM Conference on Multimedia Systems, pp. 133–144 (2011)
4. Wei, S., Swaminathan, V.: Low latency live video streaming over HTTP 2.0. In: Proceedings of the network and operating system support on digital audio and video workshop, pp. 37:37–37:42. ACM (2014)
5. Van Wallendael, G., Van Lancker, W., De Cock, J., Lambert, P., Macq, J.F., Van de Walle, R.: Fast channel switching based on SVC in IPTV environments. Broadcast. IEEE Trans. **58**(1), 57–65 (2012)
6. OpenSignal: IConnect 4G Coverage Maps. http://opensignal.com/networks/usa/iconnect-4g-coverage/ (2016). Accessed 21 Feb 2017
7. Igvita: High Performance Browser Networking. https://www.igvita.com/2014/03/26/why-is-my-cdn-slow-for-mobile-clients/ (2014). Accessed 21 Feb 2017
8. Belshe, M., Peon, R., Thomson, M., Melnikov, A.: SPDY Protocol. https://tools.ietf.org/html/draft-ietf-httpbis-http2-00/ (2012). Accessed 21 Feb 2017
9. IETF: Hypertext Transfer Protocol (httpbis). https://datatracker.ietf.org/wg/httpbis/charter/ (2012). Accessed 21 Feb 2017
10. Belshe, M., Peon, R., Thomson, M.: Hypertext Transfer Protocol Version 2. https://datatracker.ietf.org/doc/rfc7540/ (2015). Accessed 21 Feb 2017
11. Deveria, A.: Can I Use HTTP/2? http://caniuse.com/#search=HTTP%2F2 (2016). Accessed 21 Feb 2017
12. Huysegems, R., van der Hooft, J., Bostoen, T., Alface, P., Petrangeli, S., Wauters, T., De Turck, F.: HTTP/2-based methods to improve the live experience of adaptive streaming. In: Proceedings of the 23rd ACM Multimedia Conference. ACM (2015)
13. Seufert, M., Egger, S., Slanina, M., Zinner, T., Hoßfeld, T., Tran-Gia, P.: A survey on quality of experience of HTTP adaptive streaming. IEEE Commun. Surv. Tutor. **17**(1), 469–492 (2015)
14. Benno, S., Beck, A., Esteban, J., Wu, L., Miller, R.: WiLo: A rate determination algorithm for HAS video in wireless networks and low-delay applications. In: IEEE Globecom Workshops, pp. 512–518 (2013)
15. Petrangeli, S., Famaey, J., Claeys, M., Latré, S., De Turck, F.: QoE-driven rate adaptation heuristic for fair adaptive video streaming. ACM Trans. Multimed. Comput. Commun. Appl. **12**(2), 28:1–28:24 (2015)
16. Menkovski, V., Liotta, A.: Intelligent control for adaptive video streaming. In: IEEE International Conference on Consumer Electronics, pp. 127–128 (2013)
17. Claeys, M., Latré, S., Famaey, J., De Turck, F.: Design and evaluation of a self-learning HTTP adaptive video streaming client. Commun. Lett. IEEE **18**(4), 716–719 (2014)
18. Sánchez de la Fuente, Y., Schierl, T., Hellge, C., Wiegand, T., Hong, D., De Vleeschauwer, D., Van Leekwijck, W., Le Louédec, Y.: iDASH: Improved dynamic adaptive streaming over HTTP using scalable video coding. In: Proceedings of the 2nd Annual ACM Conference on Multimedia Systems, pp. 257–264. ACM (2011)
19. Akhshabi, S., Anantakrishnan, L., Dovrolis, C., Begen, A.: Server-based traffic shaping for stabilizing oscillating adaptive streaming players. In: Proceeding of the 23rd ACM workshop on network and operating systems support for digital audio and video, pp. 19–24. ACM (2013)
20. De Cicco, L., Mascolo, S., Palmisano, V.: Feedback control for adaptive live video streaming. In: Proceedings of the 2nd Annual ACM Conference on Multimedia Systems, pp. 145–156. ACM (2011)
21. Bouten, N., Famaey, J., Latré, S., Huysegems, R., Vleeschauwer, B., Leekwijck, W., Turck, F.: QoE optimization through in-network quality adaptation for HTTP adaptive streaming. In: 8th International Conference on Network and Service Management, pp. 336–342 (2012)
22. Bouten, N., Claeys, M., Latré, S., Famaey, J., Van Leekwijck, W., De Turck, F.: Deadline-based approach for improving delivery of SVC-based HTTP adaptive streaming content. In: IEEE Network Operations and Management Symposium, pp. 1–7 (2014)
23. Petrangeli, S., Claeys, M., Latré, S., Famaey, J., De Turck, F.: A multi-agent Q-learning-based framework for achieving fairness in HTTP adaptive streaming. In: IEEE Network Operations and Management Symposium, pp. 1–9 (2014)
24. Petrangeli, S., Wauters, T., Huysegems, R., Bostoen, T., De Turck, F.: Network-based dynamic prioritization of HTTP adaptive streams to avoid video freezes. In: IFIP/IEEE International Symposium on Integrated Network Management, pp. 1242–1248 (2015)

25. Schierl, T., Hellge, C., Mirta, S., Grüneberg, K., Wiegand, T.: Using H.264/AVC-based scalable video coding (SVC) for real time streaming in wireless IP networks. In: IEEE International Symposium on Circuits and Systems, pp. 3455–3458 (2007)
26. Thomas, E., van Deventer, M., Stockhammer, T., Begen, A., Famaey, J.: Enhancing MPEG DASH performance via server and network assistance. In: Proceedings of the IBC 2015 Conference (2015)
27. Latré, S., De Turck, F.: Joint in-network video rate adaptation and measurement-based admission control: algorithm design and evaluation. J. Netw. Syst. Manag. **21**(4), 588–622 (2013)
28. Badukale, S., P., W.: SPDY: An Experimental Protocol for a Faster Web. http://www.chromium.org/spdy/spdy-whitepaper/ (2009). Accessed 21 Feb 2017
29. Cardaci, A., Caviglione, L., Gotta, A., Tonellotto, N.: Performance evaluation of SPDY over high latency satellite channels. In: Personal Satellite Services, vol. 123, pp. 123–134. Springer International Publishing (2013)
30. Erman, J., Gopalakrishnan, V., Jana, R., Ramakrishnan, K.K.: Towards a SPDY'Ier mobile web? In: Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies, pp. 303–314. ACM (2013)
31. Elkhatib, Y., Tyson, G., Welzl, M.: Can SPDY really make the web faster? In: IFIP Networking Conference, pp. 1–9. IEEE (2014)
32. Wang, X., Balasubramanian, A., Krishnamurthy, A., Wetherall, D.: How speedy is SPDY? In: Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, pp. 387–399. USENIX Association (2014)
33. Chowdhury, S., Sapra, V., Hindle, A.: Is HTTP/2 more energy efficient than HTTP/1.1 for mobile users? PeerJ PrePrints **3**, e1571 (2015)
34. Müller, C., Lederer, S., Timmerer, C., Hellwagner, H.: Dynamic adaptive streaming over HTTP/2.0. In: IEEE International Conference on Multimedia and Expo, pp. 1–6 (2013)
35. Wei, S., Swaminathan, V.: Cost effective video streaming using server push over HTTP 2.0. In: IEEE 16th International Workshop on Multimedia Signal Processing, pp. 1–5 (2014)
36. Cherif, W., Fablet, Y., Nassor, E., Taquet, J., Fujimori, Y.: DASH fast start using HTTP/2. In: Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, pp. 25–30. ACM (2015)
37. van der Hooft, J., Petrangeli, S., Bouten, N., Wauters, T., Huysegems, R., Bostoen, T., De Turck, F.: An HTTP/2 push-based approach for SVC adaptive streaming. In: IEEE/IFIP Network Operations and Management Symposium, pp. 104–111 (2016)
38. Riiser, H., Endestad, T., Vigmostad, P., Griwodz, C., Halvorsen, P.: Video streaming using a location-based bandwidth-lookup service for bitrate planning. ACM Trans. Multimed. Comput. Commun. Appl. **8**(3), 24:1–24:19 (2012)
39. Zambelli, A.: IIS Smooth Streaming Technical Overview. https://www.iis.net/learn/media/on-demand-smooth-streaming/smooth-streaming-technical-overview/(2009). Accessed 21 Feb 2017
40. Jiang, J., Sekar, V., Zhang, H.: Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. IEEE/ACM Trans. Netw. **22**(1), 326–340 (2014)
41. Famaey, J., Latré, S., Bouten, N., Van de Meerssche, W., De Vleeschauwer, B., Van Leekwijck, W., De Turck, F.: On the merits of SVC-based HTTP adaptive streaming. In: IFIP/IEEE International Symposium on Integrated Network Management, pp. 419–426 (2013)
42. Sullivan, G., Ohm, J., Han, W.J., Wiegand, T.: Overview of the high efficiency video coding (HEVC) standard. IEEE Trans. Circuits Syst. Video Technol. **22**(12), 1649–1668 (2012)

Jeroen van der Hooft obtained his M.Sc. degree in Computer Science Engineering from Ghent University, Belgium, in July 2014. In August of that year, he joined the Department of Information Technology at Ghent University, where he is currently active as a Ph.D. student. His main research interest is the end-to-end Quality of Experience optimization in adaptive video streaming.

Stefano Petrangeli obtained his M.Sc. degree in Systems Engineering from University of Rome "La Sapienza", Italy, in December 2011. After joining Telecom Italia Laboratories and Polytechnic of Turin, he joined the Department of Information Technology at Ghent University in March 2013, where he is currently active as a Ph.D. student. His main research interest is the end-to-end Quality of Experience optimization in adaptive video streaming.

Tim Wauters obtained his M.Sc. degree in Electro-technical Engineering in June 2001 from Ghent University, Belgium. In January 2007, he obtained the Ph.D. degree in Electro-technical Engineering at the same university. Since September 2001, he has been working in the Department of Information Technology at Ghent University, and is now active as a post-doctoral fellow of the F.W.O.-V. His main research interests include network and service architectures and management solutions for scalable multimedia delivery services.

Rafael Huysegems obtained his M.Sc. degree in Industrial Engineering (Electronics) from Group-T Leuven, Belgium, in 1985. As a senior researcher for Nokia Bell Labs, he was actively involved in the Information Society Technologies MUSE project and the Eureka Celtic RUBENS project. His research interests include Quality of Experience, adaptive video streaming and web acceleration.

Tom Bostoen obtained his M.Sc. degree in Physics Engineering from Ghent University, Belgium, in 1998. Since then, he has been with Nokia in Antwerp, Belgium. While working for Nokia Bell Labs, he obtained the Ph.D. degree in Computer Science from the University of Leuven, Belgium, in 2014. Currently, he works as DSL software engineer at Nokia's Broadband Access business unit.

Filip De Turck leads the network and service management research group at the Department of Information Technology at Ghent University, Belgium, and iMinds. His main research interests include telecommunication network and service management, and design of efficient virtualized network systems. He is involved in several research projects with industry and academia, serves as Vice Chair of the IEEE Technical Committee on Network Operations and Management and as Chair of the Future Internet Cluster of the European Commission, and is on the TPC of many network and service management conferences and workshops.