HTTP/2-Based Adaptive Streaming of HEVC Video over 4G/LTE Networks

Jeroen van der Hooft, *Student Member, IEEE,* Stefano Petrangeli, *Student Member, IEEE,* Tim Wauters, *Member, IEEE,* Rafael Huysegems, Patrice Rondao Alface, *Senior Member, IEEE,* Tom Bostoen, and Filip De Turck, *Senior Member, IEEE*

***Abstract*—In HTTP Adaptive Streaming (HAS), video content is temporally divided into multiple segments, each encoded at several quality levels. The client can adapt the requested video quality to network changes, generally resulting in a smoother playback. Unfortunately, live streaming solutions still often suffer from playout freezes and a large end-to-end delay. By reducing the segment duration, the client can use a smaller temporal buffer and respond even faster to network changes. However, since segments are requested subsequently, this approach is susceptible to high round-trip times. In this letter, we discuss the merits of an HTTP/2 push-based approach. We present the details of a measurement study on the available bandwidth in real 4G/LTE networks, and analyze the induced bit rate overhead for HEVC-encoded video segments with a sub-second duration. Through an extensive evaluation with the generated video content, we show that the proposed approach results in a higher video quality (+7.5%) and a lower freeze time (-50.4%), and allows to reduce the live delay compared to traditional solutions over HTTP/1.1.**

***Index Terms*—HTTP Adaptive Streaming, DASH, Quality of Experience, HTTP/2, Server Push, 4G/LTE, H.265/HEVC.**

I. Introduction

TODAY, more than half of the Internet traffic is generated by video streaming applications [1]. To meet increasing requirements, the concept of HTTP Adaptive Streaming (HAS) has recently been introduced. As shown in Figure 1, content is encoded at different quality levels and temporally divided into segments with a typical length of 2 to 10 seconds. The client uses a rate adaptation heuristic to decide upon the downloaded quality for each segment, based on criteria such as the perceived bandwidth and the 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. Many heuristics and solutions have been proposed in literature, but we refer to a survey by Seufert et al. for an elaborate view on the matter [2].

Despite the many advantages of HAS, there are drawbacks as well. First, playout freezes still occur in 27% of video sessions [3]. Especially in environments with rapid bandwidth changes, the client may fail to adapt to new network conditions. Furthermore, video content is generally encoded at variable bit rate, with more bits assigned to scenes with rapid motion. As such, it often takes significantly longer to download a segment than initially estimated, increasing the chances of buffer starvation. Second, since segments of multiple seconds are typically used, the end-to-end delay in current HAS deployments is in the order of tens of seconds. This is detrimental for the QoE in live video streaming, where the delay should be as low as possible [4].

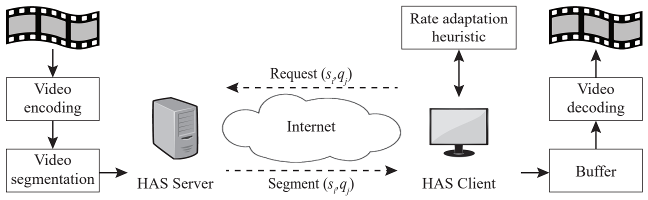

Figure 1. The concept of HTTP Adaptive Streaming.

One solution to these issues is the use of H.265/HEVC, a video compression standard which was developed to provide twice the compression efficiency of the previous standard, H.264/AVC [5]. In HEVC, coding units of up to 64x64 pixels are used instead of 16x16, and more intra-picture directions, finer fractional motion vectors and larger transform blocks are used to achieve this improvement in compression performance. Reducing the encoding bit rate has a significant impact on the QoE, as fewer data needs to be transferred from server to client. Another solution is to use segments with a sub-second duration. Shorter segments allow to limit the maximum download time of individual segments and respond faster to sudden changes in the available bandwidth. Furthermore, they allow to use a smaller buffer, which results in a potential decrease of the end-to-end delay in live streaming scenarios. Unfortunately, since every segment has to start with an Instantaneous Decoder Refresh (IDR) frame, a higher bit rate is required to achieve the same visual quality. Moreover, since a unique request is required to retrieve every single video segment, solutions with low segment duration are susceptible to high round-trip times (RTT). This problem mainly arises in mobile networks, where the RTT varies from 33 to 857 ms, depending on the network carrier and the type of connection [6].

The contributions of this letter are threefold. First, we explain an effective means to eliminate RTT cycles in Section II, using the server push feature of the recently standardized HTTP/2 protocol [7], [8]. This approach allows to effectively use short video segments, achieving the advantages described above. Second we present the details of two measurement studies in Section III. Particularly, we actively measured the available throughput in real 4G/LTE networks and performed

J. 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 project (130655) and within the EU FP7-NoE FLAMINGO project (318488). The associate editor coordinating the review of this letter and approving it for publication was B. Rong.

J. van der Hooft, S. Petrangeli, T. Wauters and F. De Turck are with Ghent University - iMinds, Department of Information Technology, Technologiepark 15, B-9052 Belgium. E-mail: jeroen.vanderhooft@intec.ugent.be.
R. Huysegems, P. R. Alface and T. Bostoen are with Nokia - Bell Labs, Copernicuslaan 50, B-2018 Antwerp, Belgium.

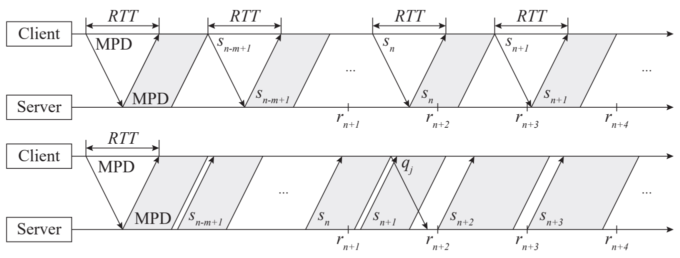

Figure 2. An example live video scenario for HTTP/1.1 (top) and HTTP/2 (bottom), where the client requests m available segments to ramp up the buffer. If the last released segment has index n, the first segment to play is $n-m+1$. Note that r_i denotes the release of segment i at server side, while s_i denotes its request for download by the client. Furthermore, quality q_j indicates that the server should change the quality of pushed segments to j.

an analysis of the induced bit rate overhead for short, HEVC-encoded video segments. Third, detailed results are presented in Section IV to characterize the gain of the proposed push-based approach compared to state-of-the-art HAS over HTTP/1.1. Final conclusions are drawn in Section V.

II. HTTP/2 Push-Based Approach

In HAS, a video 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. Based on the contents of the MPD, the client then requests video segments subsequently, generally ramping up the buffer by downloading segments at the lowest quality. After this startup phase, further decisions regarding the video quality are made by the client. 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 and bandwidth utilization in high-RTT networks. This behavior is illustrated in Figure 2, for the first phase of a live streaming session.

The HTTP/2 standard was published as an IETF RFC in February 2015, mainly focusing on the reduction of latency in web delivery [7]. Recently, a number of papers were published regarding the use of this new protocol in HAS. Wei et al. proposed a k-push approach, in which k segments are sent per request [9]. In later work, the authors proposed to change the parameter value of k dynamically based on network characteristics [10]. Focus in this research is mainly on reducing the live latency and the number of GET requests issued by the client, without considering the impact of freezes or the encoding overhead introduced by shorter video segments. In previous work, we proposed a scheme in which the base layer segments for Scalable Video Coding (SVC) are pushed by the server, while enhancement layers are pulled by the client [11]. Although a significant reduction of the freeze time is achieved compared to AVC-based solutions, the encoding overhead introduced by inter-layer dependencies makes it unfeasible to provide more than three quality representations.

In the push-based approach [8], the server uses HTTP/2's server push to push m segments to the client as soon as the MPD request is received, where m corresponds to the number of segments that fit into a preferred buffer size defined by the client. Since state-of-the-art heuristics ramp up the buffer by downloading segments at the lowest quality, it makes sense to push segments at this quality as well. As illustrated in Figure 2, 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 periodically pushes a new segment to the client at the specified quality level. Every time a segment is received, 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. Since the first m segments are pushed back-to-back when the MPD is requested, the proposed approach can significantly reduce the client's startup delay in high-RTT networks. Short segments can be used, as no RTT cycles are lost, further reducing the startup delay. Additionally, since a smaller buffer can be used, the approach allows to reduce the total end-to-end delay as well.

Preliminary evaluations showed that it is important to limit the maximum number of segments in flight; if a large amount of high-quality segments are queued in the network, e.g. right after a bandwidth drop, buffer starvation at client-side is likely to occur. An appropriate rule of thumb for the maximum number of segments k in flight is $ceil\left(\frac{RTT}{seg}\right)+1$, where k is directly proportional to the ratio of the RTT and the segment duration seg. Indeed, the higher this ratio, the more segments should be pushed in order to bridge idle RTT cycles. In our experimental setup, it will be sufficient to use $k=2$.

III. Measurement Study

A. Available Bandwidth in 4G/LTE Networks

To evaluate the proposed approach, we decided to focus on 4G/LTE networks. In order to provide a realistic evaluation, we collected throughput measurements in 4G networks within the city of Ghent, Belgium, in January and February 2016. We have built a dataset over multiple routes, measuring the available bandwidth while downloading a large file over HTTP. To guarantee appropriate download speeds, we hosted a dedicated server in iLab.t's Virtual Wall infrastructure[1], connected through a 100 Mb/s Ethernet connection. In this way, bandwidth and latency measurements indicate the performance of the wireless 4G connection, with minimal interference from the wired network. As for the client, we developed an Android application which logs all required information, running on a smartphone (Huawei P8 Lite) connected over 4G. Similar to the collection of 3G throughput traces by Riiser et al. [12], several properties are logged, among which the GPS coordinates, the number of bytes received since last datapoint and the number of milliseconds since last datapoint. From these last two entries, the average throughput can be obtained.

We collected throughput logs for six types of transportation: foot, bicycle, bus, tram, train and car[2]. As an example, Figure 3 shows the selected route in a car and the measured bandwidth over time. Lower throughput values are observed when connectivity is limited, due to tunnels, large buildings and bad coverage in general. Also, the type of transportation and the selected route have a strong impact on the available bandwidth. As an example, the average throughput on a train around the city was 22.8 Mb/s ± 14.6 Mb/s, while this was

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

[2] The authors would like to thank T. Baele and L. Timperman for their kind assistance during the data collection.

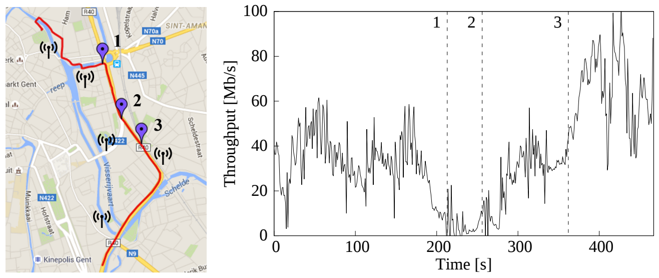

Figure 3. A car travelling from north to south (left), along with the measured throughput (right). When travelling from (1) to (2), large townhouses on the right side impede the client's connection. Arriving at (2), the client switches to a new antenna with better coverage. Once an open area is reached in (3) and a new antenna is again selected, throughput improves significantly.

33.9 Mb/s ± 15.8 Mb/s in a car driving on the ring road. The measured bandwidth ranged from 0 Mb/s (connection interrupted) through 111 Mb/s (higher than 100 Mb/s because of network queuing), with an average of 30.3 Mb/s ± 16.7 Mb/s. The complete dataset, which consists of 40 traces and covers 5 hours of monitoring, has been made available online [13].

B. HEVC-Encoded Video

In this research, we decided to focus on HEVC because of its promising compression efficiency. Since our intention is to use video segments with a sub-second duration, it is important to analyze the induced encoding overhead. The considered video sequence in our analysis and evaluation is Netflix's El Fuente, which has a total length of 476 seconds and a frame rate of 60 FPS. The video is encoded using HEVC, providing six quality levels at nominal bitrates of 0.3, 1.0, 2.3, 5.2, 10.9 and 21.4 Mb/s, with a spatial resolution ranging from 540p to 2160p video. Using the x265 encoder[3], the video is segmented using five segment durations: 133, 267, 500, 1000 and 2000 ms. To allow each segment to be decoded independently, every segment starts with an IDR frame and the Group Of Pictures (GOP) length is set to 8, 16, 30, 60 and 120 frames respectively. To assess the impact of shorter GOP lengths on the compression performance, the encodings for different segment durations have been set to target the same visual quality and allow a subsequent overhead in the achieved nominal bitrate. To realize this, we have selected the Constant Rate Factor (CRF) rate control implemented in the x265 encoder. 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 0.233 dB. Compared to a GOP length of 120 frames, the average over-head is 6.3%, 9.2%, 29.3% and 60.5% for a GOP length of 60, 30, 16 and 8 frames respectively. Figure 4 shows the obtained bit rates of the six quality representations, with a clear increase for segments with a sub-second duration. In the next section, the proposed approach will be evaluated for a segment duration of 500 ms. This allows to reduce the buffer size to the order of seconds and increase video quality in high-RTT networks, while the overhead is limited to 9.2%.

As for the encoding time, using a multicore platform with Intel Core i7 CPUs and an Nvidia GTX 980 GPU, x265 with

[3] http://x265.org

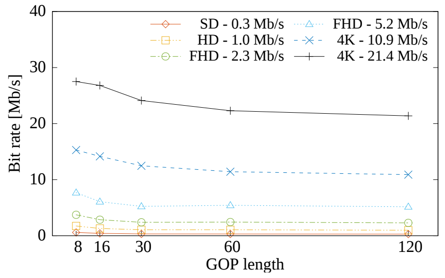

Figure 4. Obtained video bit rates for the different quality representations and a GOP length of 8, 16, 30, 60 and 120 frames.

OpenCL acceleration was able to encode the FHD content in real-time, with frame rates ranging from 63 FPS (GOP 8) to 68 FPS (GOP 120). For the 4K representations however, frame rates ranged from 21 FPS (GOP 8) to 24 FPS (GOP 120). Faster software HEVC encoders were reported recently to be able to encode 4K in real-time on similar CPU platforms [14].

IV. Evaluation

A. Experimental Setup

To allow a fair comparison of the proposed approach with traditional HAS, a network topology is emulated using the MiniNet framework[4]. It consists of a single client, streaming the encoded video from a dedicated Jetty web server[5]. A new request handler is defined, which processes the client's GET requests using a specific query to start the pushing of segments at a given quality representation. The client is implemented on top of the libdash library[6], the official reference software of the MPEG-DASH standard. We provided support for HTTP/2 using the nghttp2 library[7], and implemented the required logic to asynchronously handle pushed video segments. Client-side rate adaptation is based on the FINEAS heuristic by Petrangeli et al. [15]. This heuristic estimates the segments' download time to achieve a target buffer filling level, resulting both in a higher video quality and a lower amount of playout freezes compared to state-of-the-art solutions. To avoid an excessive amount of quality switches for short segments, the client is only allowed to increase the quality every 2 s. The collected 4G traces for same-type vehicles are merged together, in order to obtain 30 unique bandwidth traces with a minimal length of 494 s and an average bandwidth of 30.3 Mb/s ± 16.8 Mb/s. Using traffic control command *tc* for traffic shaping, the client can stream 30 episodes of the video with a different bandwidth pattern for every episode. A lower threshold of 50 kb/s is used, in order to guarantee correct packet scheduling with *tc*. The bandwidth at server-side is fixed at 100 Mb/s, same as in the measurement study.

B. Obtained Results

First, the performance of traditional HAS and the push-based approach are evaluated for increasing values for the RTT, with an initial buffer size of 10 s. Note that when playout freezes occur, the buffer is expanded as to hold all segments released at server-side. Figure 5 shows that for HTTP/1.1, the video quality, averaged out over all segments - 0 for the

[4] http://mininet.org/
[5] http://www.eclipse.org/jetty/
[6] https://github.com/bitmovin/libdash
[7] https://nghttp2.org/

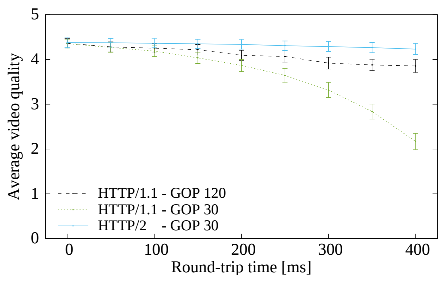

Figure 5. Impact of the RTT on the video quality, both for HTTP/1.1 and HTTP/2 with an initial buffer size of 10 seconds.

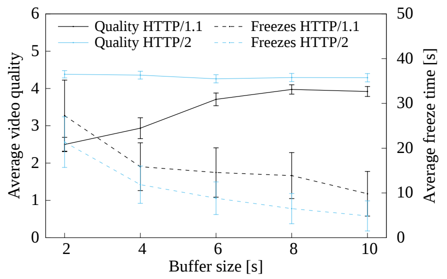

Figure 6. Impact of the buffer size on the video quality and freeze time, both for HTTP/1.1 (GOP 120) and HTTP/2 (GOP 30) with an RTT of 300 ms.

lowest quality level, 5 for the highest - drops significantly for higher RTTs, regardless whether a segment duration of 2000 or 500 ms is used. The video quality for the proposed approach over HTTP/2 is not impacted however, because bandwidth utilization is maximized by actively pushing segments from server to client. Short segments can thus effectively be used, which is not true for traditional HAS over HTTP/1.1.

In a second set of experiments, performance is evaluated as a function of the initial buffer size, for an RTT of 300 ms. Figure 6 shows that, while the video quality over HTTP/1.1 increases for larger values of the buffer size, it is more or less constant for the push-based approach. Despite an encoding overhead of 9.2%, the average quality is significantly higher because of better bandwidth utilization. As for the freeze time, a clear decrease is observed for higher buffer sizes, because a playout freeze is less likely if more content can be buffered at client-side. More importantly however, the freeze time for the proposed approach is always lower than for traditional HAS, because the client can respond faster to changes in the available bandwidth or buffer fulling.

The most relevant results are summarized in Table I. For a standard buffer size of 10 s, the proposed approach results in a significantly higher video quality (+7.5%), a lower freeze time (−50.4%) and a lower startup delay (−25.0%) compared to traditional HAS. Focusing on a reduction of the live delay, a smaller buffer size of 6 s with pull-based HAS results in a significantly lower video quality (−3.4%) and a higher freeze time (+54.7%), compared to a buffer size of 10 s. However, comparing results for the push-based approach and a buffer size of 6 s, with traditional HAS and a buffer size of 10 s, a higher video quality (+7.1%) and a lower startup delay (−25.3%) are obtained, while differences for the freeze time are not statistically significant (two-tailed Wilcoxon signed-rank test, $p = 0.82$). This shows that the proposed approach allows the client to follow the live signal more closely, without losing performance on other metrics.

HTTP	Buffer [s]	Video quality	Quality switches	Freeze time [s]	Startup delay [s]
HTTP/1.1	10	4.919 ± 0.132	49.633 ± 6.663	9.817 ± 4.988	2.408 ± 0.052
HTTP/1.1	6	4.754 ± 0.140	64.333 ± 7.254	15.190 ± 5.204	2.405 ± 0.047
HTTP/2	10	5.288 ± 0.111	52.067 ± 9.766	4.867 ± 3.361	1.806 ± 0.085
HTTP/2	6	5.270 ± 0.117	60.233 ± 10.600	8.977 ± 4.363	1.799 ± 0.084

Table I
PERFORMANCE SUMMARY FOR AN RTT OF 300 ms. AVERAGE VALUES ARE REPORTED, ALONG WITH THE 95% CONFIDENCE INTERVALS.

V. CONCLUSIONS

In this letter, we discussed an HTTP/2 push-based approach for HTTP Adaptive Streaming (HAS) which enables the use of video segments with a sub-second duration in mobile, high round-trip time networks. We quantified the encoding overhead for short HEVC-encoded segments, and determined that the segment duration should not be lower than 500 ms to limit the overhead to 9.2%. We also performed measurements for the available bandwidth in real 4G/LTE networks within the city of Ghent, Belgium, and created a dataset which has been made available online. Using the encoded content and collected throughput traces in an extensive evaluation, we showed that the presented approach results in a higher video quality (+7.5%) and a lower freeze time (−50.4%), and allows to reduce the live delay compared to solutions over HTTP/1.1. Future work will focus on further improving the user's QoE through HTTP/2 features such as request/response multiplexing and stream prioritization, on reducing the encoding overhead for short video segments and on adaptively changing the segment duration based on network conditions.

REFERENCES

[1] Sandvine Incorporated, "Global Internet Phenomena Report," 2016.
[2] M. Seufert *et al.*, "A Survey on Quality of Experience of HTTP Adaptive Streaming," *IEEE Communications Surveys Tutorials*, vol. 17, no. 1, pp. 469–492, 2015.
[3] Conviva, "Viewer Experience Report," 2015.
[4] T. Lohmar *et al.*, "Dynamic Adaptive HTTP Streaming of Live Content," in *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2011, pp. 1–8.
[5] G. J. Sullivan *et al.*, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.
[6] OpenSignal, "IConnect 4G Coverage Maps," 2014. [Online]. Available: http://opensignal.com/networks/usa/iconnect-4g-coverage
[7] M. Belshe *et al.*, "Hypertext Transfer Protocol Version 2," RFC Editor, Tech. Rep. Internet-Draft, 2015. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-httpbis-http2/
[8] R. Huysegems *et al.*, "HTTP/2-Based Methods to Improve the Live Experience of Adaptive Streaming," in *ACM Multimedia Conference*, 2015, pp. 541–550.
[9] S. Wei *et al.*, "Low Latency Live Video Streaming over HTTP 2.0," in *ACM Network and Operating System Support on Digital Audio and Video Workshop*, 2014, pp. 37:37–37:42.
[10] M. Xiao *et al.*, "Evaluating and Improving Push-Based Video Streaming with HTTP/2," in *ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2016, pp. 3:1–3:6.
[11] J. van der Hooft *et al.*, "An HTTP/2 Push-Based Approach for SVC Adaptive Streaming," in *IEEE/IFIP Network Operations and Management Symposium*, 2016, pp. 104–111.
[12] H. Riiser *et al.*, "Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications," in *ACM Conference on Multimedia Systems*, 2013, pp. 114–118.
[13] J. van der Hooft *et al.*, "4G/LTE Bandwidth Logs," 2016. [Online]. Available: http://users.ugent.be/~jvdrhoof/dataset-4g/
[14] T. K. Heng *et al.*, "A Highly Parallelized H.265/HEVC Real-Time UHD Software Encoder," in *IEEE International Conference on Image Processing*, 2014, pp. 1213–1217.
[15] S. Petrangeli *et al.*, "QoE-driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming," *ACM Trans. on Multimedia Computing, Communications and Applications*, vol. 12, no. 2, pp. 28:1–28:24, 2015.