A Web-Based Framework for Fast Synchronization of Live Video Players

Dries Pauwels*, Jeroen van der Hooft*, Stefano Petrangeli*, Tim Wauters*, Danny De Vleeschauwer†, and Filip De Turck*
*Ghent University - iMinds, Department of Information Technology, Technologiepark 15, B-9052 Ghent, Belgium
†Nokia - Bell Labs, Copernicuslaan 50, B-2018 Antwerp, Belgium
E-mail: dries.pauwels@intec.ugent.be

***Abstract*—The increased popularity of social media and mobile devices has radically changed the way people consume multimedia content online. As an example, users can experience the same event (e.g. a sports event or a concert) together using social media, even if they are not in the same physical location. Moreover, the introduction of the HTTP Adaptive Streaming principle has made it possible to deliver video over the best-effort Internet with consistent quality, even for mobile devices. One of the challenges within this context is the synchronization of multimedia playback among geographically distributed clients. To solve this issue, we propose a Web-based framework which allows to synchronize the playback of different clients. We also present a novel hybrid approach for adaptive streaming to allow fast synchronization among different clients, which relies on HTTP/2's server push feature in combination with sub-second video segments.**

In this paper, we detail the proposed framework and provide a comprehensive analysis of its performance. Experiments show that the novel hybrid approach can reduce synchronization time with 19.4% compared to standard adaptive streaming over HTTP/1.1 when bandwidth is limited to 2.5 Mb/s and an RTT of 150 ms. The gain increases even more when a higher throughput is available. The obtained results entail that the proposed framework can provide quality of experience for all users watching online video together.

***Index Terms*—Video Synchronization, Inter-Destination Multimedia Synchronization, HTTP Adaptive Streaming, MPEG-DASH, HTTP/2, HTML5**

I. Introduction

Multimedia applications have become more and more important, with video traffic accounting for 70% of the consumer Internet traffic today. By 2020, this quota is even expected to increase to 82% [1]. HTTP Adaptive Streaming (HAS) is the most used technology to stream video over the best-effort Internet. In HAS, video content is temporally split into segments and encoded at several quality representations. At the start of a video session, the client requests the Media Presentation Description (MPD) of the video, which describes the various spatial, temporal and quality dimensions of the content as well as the different representations. The client is equipped with a rate adaptation heuristic, which allows it to adapt to varying network conditions during the playback by requesting segments at a different quality. When the available bandwidth drops for instance, the client can select a lower quality level in order to prevent buffer starvation and consequently avoid a freeze in the video playout.

Meanwhile, the popularity of smart-TVs, smartphones and tablets has changed the way multimedia content is enjoyed. As an example, a current trend is for people to watch multimedia content together although not necessarily in the same physical location. Imagine a group of friends following the same football match on different devices in different locations as shown in Figure 1. If we assume that they are communicating over voice or text chat and one of them can see a goal before the others, this results in a bad Quality of Experience (QoE). One of the challenges within this context is therefore the synchronization of multimedia playback among these geographically distributed clients, which is referred to as Inter-Destination Multimedia Synchronization (IDMS). Geerts et al. have shown that users communicating via voice or text chat start to notice differences above 500 ms in delay in synchronization [2].

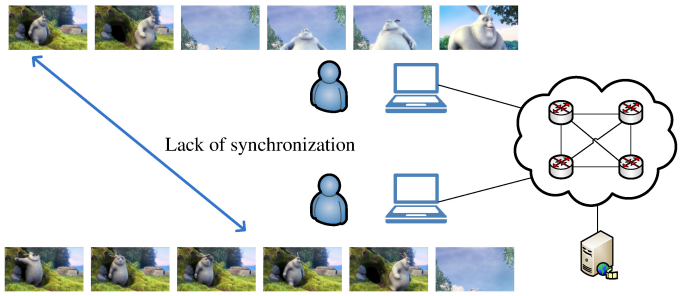


Fig. 1: Synchronization problem when two users watch the same content in different locations

Most of the devices which are currently connected to the Internet, share a common characteristic, i.e. they are equipped with a web browser. Recent browsers use HTML5 for the playback of media. With this technology, it becomes possible to embed media in web pages without the need of plug-ins (such as Flash). Instead, an HTMLVideoElement is introduced, which can be extended with a useful API. The DASH-IF Reference Player[1], a JavaScript implementation of the MPEG-DASH standard, uses the HTMLVideoElement for playing HAS streams in a browser.

In order to solve the issue of IDMS, we introduce a browser-based framework to enable the synchronization of different HAS clients. This approach enables clients to synchronize the playback of both Video On Demand (VOD) and live streams. The clients streaming the same video are grouped into a synchronization group. Among these clients, one is chosen to be the master client. All clients will start the playout of the video at the same position as the master client.

[1]https://github.com/Dash-Industry-Forum/dash.js/

In order to reduce the start-up delay and the synchronization time among different clients, we propose a hybrid solution that uses different segment durations for different quality representations. Particularly, the lowest quality level of the video is provided in a super-short segments version. With super-short segments, we define segments with a segment duration shorter then one second. When a client needs to synchronize its playout with the reference signal (e.g. at start-up or after a freeze or pause), the lowest quality in the super-short segments version is requested.

Another important aspect of the proposed framework is that the clients automatically adjust their playback rate in order to reach synchronization with the playback position of the master. We introduce an Adaptive Media Playout (AMP) algorithm that aims to smoothly increase the playback rate of the video to avoid a video freeze while minimizing the time needed to synchronize the video playout. This way, the use of the AMP principle allows to maximize the users' QoE.

The remainder of this paper is structured as follows. Section II gives an overview of the related work on IDMS, AMP, HAS over HTTP/2 and dynamic segment duration in HAS. The proposed framework is presented in Section III. An evaluation of the framework is discussed in Section IV, while Section V concludes the paper.

II. Related Work

A. Inter-Destination Multimedia Synchronization

Most of the previous solutions for IDMS use the Real-time Transport Protocol (RTP) in combination with the Real-time Transport Control Protocol (RTCP) to achieve synchronization [3]. Nowadays, the HAS principle is preferred over RTP for the delivery of multimedia content over the best-effort Internet [4]. With pull-based solutions such as HAS, the mechanisms for synchronization can be moved to the client-side, which reduces the complexity and increases scalability of the video delivery and synchronization systems.

Jung et al. introduced a Web-Based IDMS framework for MPEG-DASH in which one client is selected to be the synchronization manager of the synchronization group [5]. This synchronization manager notifies the playback position to the other clients. The disadvantage of this approach is that the synchronization accuracy depends on the network delay of the synchronization interface. Another difference with our framework is that the playback of the clients in the same synchronization group, is started when all the clients are able to start their playback. Also, when a client experiences a freeze, every client in the synchronization group is paused. Rainer et al. proposed a self-organized distributed framework for adaptive media streaming [6]. The MPD-file is extended with information about the other clients in the synchronization. With this information, the clients are able to negotiate a common playback position to which they synchronize. Compared to this distributed control scheme, we use a master-slave approach, where the slaves in the synchronization group use the playback position of the master as an anchor point. By introducing a *Synchronization Server*, the clients do not have to be aware of each other. This reduces the amount of connections needed to achieve synchronization among the clients compared to a purely peer-to-peer approach.

B. Adaptive Media Playout

Adaptive Media Playout (AMP) was originally used to avoid buffer underflows and overflows. AMP allows the client to buffer less data and, thus, introduces less delay to reach a given playback reliability [7], [8]. Instead of skipping or pausing the content to achieve a certain synchronization point, Montagud and Boronat proposed to use AMP to acquire an overall synchronization between distributed clients [9]. Recently, an AMP algorithm was introduced that takes the visual and acoustic features of the content into account so that the probability of a user noticing the change in the playback rate is low [10]. This algorithm assumes that the clients can easily subtract the visual and acoustic features from the content that is being played, which cannot be taken for granted. In Section-III-D we introduce an AMP-algorithm that takes into account the current buffer filling level and an estimation of the segment download time to minimize the synchronization time and avoid video freezes.

C. HAS over HTTP/2

In February 2015, the new HTTP/2 standard was published as an IETF RFC [11]. HTTP/2 is based on Google's SPDY protocol, whose main goal was to improve the loading time of web pages [12]. HTTP/2 includes new features such as request-response multiplexing, header compression and server push to improve the performance of standard HTTP. Mueller et al. were the first to investigate HTTP/2 in a video streaming scenario [13]. The existing standard HTTP layer is replaced with SPDY. They show that, due to the overhead that gets introduced due to Secure Socket Layer (SSL) and the framing of SPDY, the benefits of using header compression, a persistent connection and pipelining are almost completely neglected. Wei et al. explored how new HTTP/2 features can be used to improve HAS [14]. By using the server push feature in HTTP/2, they are able to avoid the request explosion problem while lowering latency by reducing the segment duration. A disadvantage to this approach is that, when the client switches to a new quality, there are still segments being pushed at the previous quality. Fablet et al. introduced DASH fast start, which lowers the startup delay of a DASH video [15]. Huysegems et al. presented ten novel HTTP/2-based methods to improve the QoE of HAS [16]. Recently, van der Hooft et al. introduced an HTTP/2 push-based approach for Scalable Video Coding (SVC) adaptive streaming where only the base layer of the SVC-encoded video is pushed from server to client [17]. This solution allows to eliminate one RTT cycle for every video segment, which has a significant impact on the user's QoE. However, a limited amount of quality representations can be used because of SVC's encoding overhead.

D. Dynamic Segment Duration in HAS

In standard HAS, the client reacts to varying network conditions by changing the requested video quality. More recently, the use of variable segment duration for HAS has emerged. By using dynamic segment duration instead of fixed segment duration, Yun and Chung show that a seamless playback can be provided while reducing the buffering delay [18].

By using super-short segments, a lower start-up delay and a lower synchronization time for the clients can be achieved.

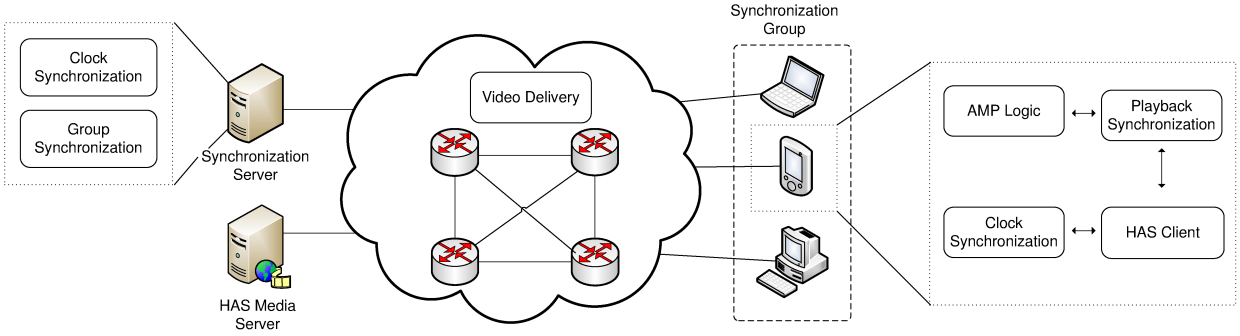


Fig. 2: The architecture of the proposed framework

Super-short segments however introduce an encoding overhead. In HAS, every segment starts with an Instantaneous Decoding Refresh (IDR) frame in order to be decoded independently of other segments. For video coding standards such as H.264 Advanced Video Coding (AVC) and H.265 High Efficiency Video Coding (HEVC), the encoding efficiency of an IDR frame is significantly lower than the efficiency of frames composed of P or B slices. As a result, a higher bit rate is needed to reach the same visual quality as for longer segments. To overcome this issue, Consequently, in order to reduce the encoding overhead, super-short segment are only used for the lowest quality representation. As most rate adaptation heuristics start streaming at the lowest quality, this choice allows to reduce the start-up and synchronization time, while maintaining the overhead limited.

III. Overview of the Proposed Framework

Our proposed framework consists of four essential components, discussed in four subsections below. As shown in Figure 2, each client first needs to synchronize its local time to common wallclock. This clock synchronization is discussed in Section III-A. Next, the client joins a synchronization group to synchronize a video session as explained in Section III-B. The amount of time it takes the clients to synchronize to a reference signal strongly depends on the method used to deliver the video. In Section III-C, we therefore present a hybrid method using HTTP/2 which enables clients to synchronize faster with the master. Finally, as part of the playback synchronization, a new AMP logic is described in Section III-D, which aims to carry out the process of adjusting the media playback rate at the clients to further reduce synchronization times. Figure 3 and 4 show the messages that are exchanged at the beginning of a video session, right after the web page (where the video is embedded in), is loaded until the video playback is synchronized.

A. Clock Synchronization

In order for the clients to synchronize their playback, they first have to share a common wallclock. Although the internal clock of most devices connected to the Internet are synchronized by using the Network Time Protocol (NTP), accurate clock synchronization cannot be guaranteed due to clock drift or a firewall blocking the service. Arntzen and Borch propose a new programming model for precisely timed Web applications [19]. Timing objects located at the clients are synchronized with a timing object hosted online by a

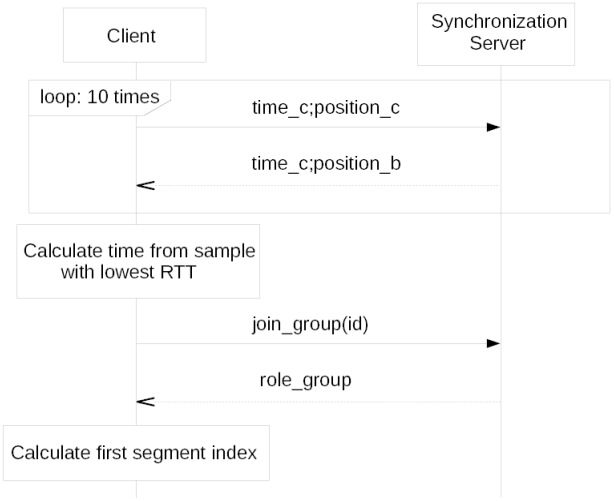


Fig. 3: Sequence diagram of the messages between a client and the Synchronization Server

specialized service. This concept is based on the Media State Vector (MSV), a multi-device timing mechanism which allows timed operations across Web pages hosted by different devices [20]. An important condition for the synchronization is that the clock values are increased with a constant rate. For this reason, the used clock has to be independent of the system time, which might be adjusted manually or skewed by software like NTP. Apart from the system clock, the clients keep a timer, the High Resolution Timer (HRT) API [2], which is included in the major Web browsers and is not skewed when the system time changes. It is the position of this timer that will be synchronized with the servers timer.

B. Group Synchronization

A synchronization group consists of clients whose playback has to be synchronized to a given synchronization point. The way that the clients are divided in synchronization group can be expanded to multiple use cases, where group selection is (1) performed by the clients, setting a group id in the query of the web page, (2) handled by the Synchronization Server based on the played content, or (3) done by the Synchronization Server based on the clients' geographical locations. Among the clients in a synchronization group, one client is selected to be the master client to whom the other clients, so called slave clients,

[2]http://w3c.github.io/hr-time/

will synchronize. In all cases described above, the client who joins the synchronization group first is selected as the master. The master's only task is to provide a reference signal to the Synchronization Server for clients who will join the group in the future. As such, when a master leaves a group, this does not affect other clients in the group. Clients belonging to the same synchronization group communicate directly to a Synchronization Server. The exchanged messages between a slave client and the Synchronization Server are illustrated Figure 3. When a new client wants to join a synchronization group, it sends a *join_group*-message to the Synchronization Server consisting of a *group_id*. The server answers with a *role_group*-message indicating whether the client is a master or a slave in the synchronization group. In the former case, the client can start the video session without synchronization. As soon as the master has started its video playout, it sends a *sync_reference*-message to the Synchronization Server. This message contains the index number of the first segment the master played and the timestamp indicating when the playout started. Both the timestamp and the index are used as a reference signal for the entire synchronization group. In case the client is a slave, the Synchronization Server also returns a *role_group*-message, which now includes the reference index and the reference timestamp of the master client. For ease of use, we assume that the address of the Synchronization Server is known by the clients. In future work, the address could be stored within the MPD. The playback synchronization is handled by the slave clients themselves. Based on the reference index and timestamp of the master, the slave clients calculate which segment the master is currently playing. The first segment a slave client requests to the HAS media server, is then the next segment the master is going to play. If the slave client receives this segment before the master has started the playback of the segment, the client waits and starts the playback of the segment at the same moment as the master. If the segment is received after the master has started playing it, then the slaves apply a new AMP logic to increase the playback rate and reach synchronization with the master. This logic is explained in Section III-D.

C. Video Delivery

Traditionally, HTTP/1.1 is used to deliver video segments from a HAS media server to a client. As an alternative to achieve faster synchronization, we propose a hybrid approach which relies on HTTP/2's server push feature, in combination with super-short video segments.

In order for a client to start the playback of a video, the video buffer filling level has to exceed a certain threshold. The time it takes for a slave client to synchronize to the reference signal therefore strongly depends on the amount of time it takes for the first segments to be fetched. Using a short segment duration can strongly reduce the synchronization time, since short segments can be downloaded faster. A lower segment duration however results in a significant increase of the number of HTTP GET requests, which would in turn increase the load on both the network and the HAS server. Also, this approach is susceptible to large RTTs: because segments are requested sequentially, an RTT is lost between subsequent transfers. By using the push feature of the HTTP/2 protocol,

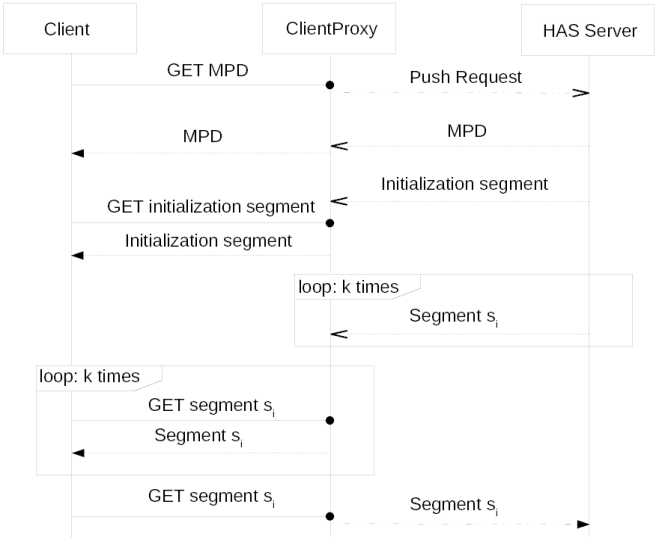


Fig. 4: Sequence diagram of the messages between a client, the Client Proxy and the HAS Server

the HAS client can request multiple segments at once to avoid lost RTT cycles between subsequent retrievals, resulting in a lower start-up delay [16]. In the proposed approach, the lowest video quality representation is provided with super-short video segments. Because every segment has to start with an IDR frame in order to be decoded independently from other segments, the encoding efficiency of an IDR frame is significantly lower when using short segments compared to longer segments. As a result, a higher bit rate is needed to reach the same visual quality as for longer segments. This is why we chose to only use super-short segments for the lowest video quality.

There are however some issues related to server push feature. The Web clients are not aware of the segments that are already pushed by the HAS server, which can cause the client to request a segment that is being pushed by the server but not yet received, leading to waist of bandwidth. That is why, similar to work by Fablet et al., we place a Client Proxy between the client and the HAS server that is able to cache the pushed segments [15]. The Client Proxy is also able to hang a request from the client for a pushed request that did not arrive yet and forward it when it does arrive. As soon as the client knows the index of the first segment to be played , it sends a request for the MPD to the Client Proxy indicating to start pushing segments starting from that index, over HTTP/1.1. The Client Proxy forwards this request as a push request to the HAS server over HTTP/2. Also mentioned in this request is the amount of segments k that the server needs to push to the Client Proxy. In order to switch to a larger segment duration when the pushed segments have arrived, the end time of the last segment should be the start time of a large segment. Therefore, the amount of segments to push is calculated as $k + \delta$. where k is a fixed amount of segments and δ is the amount of short segments needed to complete a long segment. The MPD, the initialization segment of the lowest quality representation and the first $k + \delta$ super-short segments are pushed from the HAS server to the Client Proxy

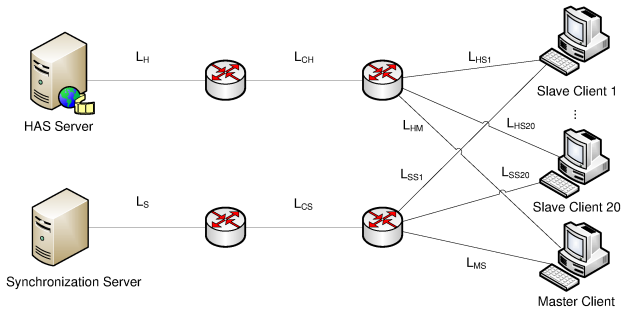


Fig. 5: Experimental setup

using HTTP/2 push [21]. This way, the client is able to start its playback faster and consequently reduce the synchronization time. Once the first $k+\delta$ segments are fetched, the client can request the following segment with a longer segment duration.

D. AMP Logic

When a slave client is not in sync with the master, either at the start of a video session or when a freeze or a pause occurred, the client increases the playout rate to synchronize with the master. As an initial proposition, the playout rate is then doubled in order to both reduce the synchronization time, and avoid depleting the video player buffer. In order to prevent a buffer starvation after or during fast playout, we introduce a buffer panic threshold B_{tr}. If the buffer level at time t is $B(t)$ and the difference in time with the master at time t is $\Delta T(t)$, we define the following condition:

$$B(t) > B_{tr} + 2 * \Delta T(t) \quad (1)$$

If this condition is met, we double the playout rate to reach synchronization. If the condition is not met, we wait until the next segment is fetched and check condition (1) again.

IV. Evaluation and Discussion

A. Experimental Setup

In our evaluation, a part of a football match is used. The video sequence has a total length of 620 seconds with a frame rate of 24 FPS. It is encoded in variable bitrate with H.264/AVC using five different quality representations, with nominal bit rates of 700, 1200, 1800, 2900 and 4600 kb/s for segments with a segment duration of 2000 ms. The video is provided with a segment duration of both 500 and 2000 ms. The average encoding overhead for 500 ms segments is 17.5%. With our hybrid approach, the lowest quality representation is segmented with both 500 ms and 2000 ms, the other representations use segments of 2000 ms. Our experiments are performed on the iMinds iLab.t Virtual Wall platform[3]. The Virtual Wall facilities consist of 100 nodes (dual processor, dual core servers) interconnected via a non-blocking 1.5 Tb/s VLAN Ethernet switch. The network topology is shown in Figure 5. It consists of twenty clients streaming video from a HAS Server. The clients are connected with a Synchronization Server to enable the synchronization of their video playback. On link L_{CS}, the bandwidth is limited to 1 Gb/s and the RTT is fixed to 50 ms. The bandwidth on link L_{CH} is limited to 100 Mb/s and several values of the RTT are tested. On links L_{HM} and L_{HS}, the bandwidth is equally limited to several values of throughput to evaluate the performance of the proposed framework in different network scenarios.

The clients are implemented on top of the DASH-IF Reference Player, a JavaScript implementation of the MPEG-DASH standard[4]. This code has been extended to implement the proposed synchronization framework. WebSocket functionality[5] was added to the clients in order to communicate with the Synchronization Server. Clients run the web page where the video is embedded on a Google Chrome browser (v50.0.2661.75) in headless mode. The Client Proxy is implemented on top of NodeJS[6]. For HTTP/2, a NodeJS implementation is used[7]. The Client Proxy acts as an HTTP/1.1 server towards each client and an HTTP/2 client towards the HAS server.

The HAS media server is based on the Jetty web server[8] and was extended to provide support for HTTP/2. When the server receives a push request, k consecutive segments are pushed to the client over HTTP/2. The HAS media server also stores the web page on which the video is embedded. The Synchronization Server is a Jetty-based WebSocket server. It selects a synchronization group for a new client depending on the URL of the MPD, which is composed in the *join_group*-message. Hence, clients which request the same MPD, belong to the same synchronization group.

The external timing object to synchronize the clocks of the clients (cf. Section 3-A) is written in Python and the communication with the video clients is carried out over the WebSocket protocol. The external timing object is co-located with the Synchronization Server.

B. Evaluation Metrics

We compare the performance of the proposed HTTP/2-based approach with HTTP/1.1. First, we compare the synchronization time, defined as the time between the loading of the web page and the moment at which the slave client is synchronized with the master client, expressed in seconds. Second, the average video quality, expressed in a number between 0 and 4. We also evaluate the impact of the proposed approaches on the synchronization accuracy. Finally, we compare the impact of different values of k on the synchronization time. The considered video trace is streamed five times for every configuration. The results are shown using the averages of the twenty slave clients and the 95% confidence intervals.

C. Evaluation Results

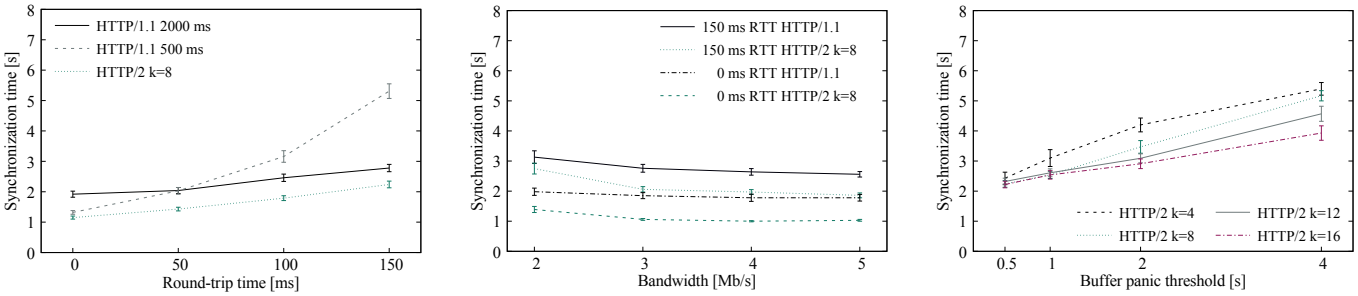
Figure 6a shows the impact of the RTT on the synchronization time using HTTP/1.1 with a segment duration of 500 ms and 2000 ms and our HTTP/2 hybrid approach where the value of k is set to 8 segments. The buffer panic threshold is set to 0.5 s for all approaches. The synchronization time is similar for HTTP/1.1 with a segment duration of 500 ms and HTTP/2, when the RTT is negligible. In this case, there is no

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

[4]http://mpeg.chiariglione.org/standards/mpeg-dash
[5]https://websocket.org/
[6]https://nodejs.org
[7]https://github.com/molnarg/node-http2
[8]https://webtide.com

TABLE I: A summary for the different approaches, comparing the results for an RTT of 150 ms, bandwidth of 2.5 Mb/s and a buffer panic threshold of 0.5 s.

HTTP	Segment duration [ms]	Startup time [s]	Synchronization time [s]	Accuracy [s]	Average video quality
HTTP/1.1	2000	2.11 ± 0.03	2.78 ± 0.12	0.011 ± 0.001	1.95 ± 0.00
HTTP/1.1	500	2.02 ± 0.02	5.31 ± 0.24	0.010 ± 0.001	1.40 ± 0.00
HTTP/2 k=8	hybrid	1.66 ± 0.03	2.24 ± 0.11	0.009 ± 0.002	1.95 ± 0.00



(a) (b) (c)

Fig. 6: Impact of the RTT, bandwidth and buffer panic threshold on the synchronization time. Unless the values are variable, the bandwidth is fixed to 2.5 Mb/s, the RTT is 150 ms and the segment duration is 2000 ms.

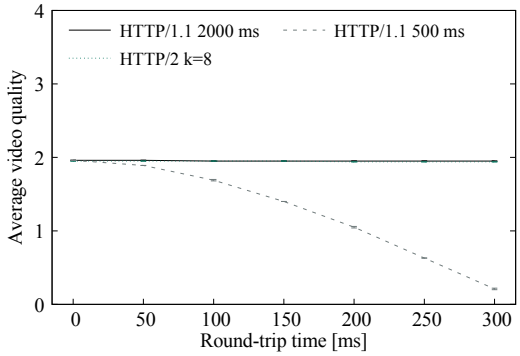


Fig. 7: Impact of the RTT on the average video quality, for HTTP/1.1 with a segment duration of 500 ms and 2000 ms and HTTP/2. The bandwidth is fixed to 2.5 Mb/s.

difference between pushing and pulling the segments. When the RTT increases, this will influence the startup time. As a result, it will take more time for the client's buffer filling level to meet condition (1) and thus increase its playout rate.

Figure 6b shows the impact of an increasing bandwidth on the client's synchronization time, for both a negligible RTT and an RTT of 150 ms comparing HTTP/1.1 with a segment duration of 2000 ms and our HTTP/2 approach. Both for a negligible RTT and an RTT of 150 ms, our HTTP/2 is able to synchronize its video playout faster. Because the encoding overhead associated with super-short segments has a higher impact when the available bandwidth is small, the difference between the two methods increases with an increasing bandwidth. In Figure 6c, the impact of an increasing buffer panic threshold is shown for HTTP/2 where different values of k are evaluated. For a panic threshold of 0.5 s, the synchronization time is higher for k equals 4 segments, because the condition for the client to increase its playout rate will not yet be met after the first four segments are loaded. For an increasing buffer panic threshold, more segments need to be loaded to meet this condition. That is why higher values for k have a lower synchronization time.

In Figure 7, finally, the impact of the RTT on the average video quality is shown. When the RTT increases, the average video quality for HTTP/1.1 with a segment duration of 500 ms is significantly lower compared to the other methods. Using HTTP/2, the average quality is 1.95 for an RTT of 300 ms. Bandwidth utilization is significantly higher in this case because the first $k+\delta$ segments are pushed by the HAS server. With this approach, the average video quality is more or less similar to HTTP/1.1 with a segment duration of 2000 ms.

A summary is shown in Table I for all approaches with a bandwidth of 2.5 Mb/s, an RTT of 150 ms and a panic threshold of 0.5 s. We conclude that the different approaches do not have an impact on the synchronization accuracy, which has an average value of 10 ms. If we compare the startup time for HTTP/1.1 with a segment duration of 500 ms and HTTP/2, a reduction of 21.3% is observed. This means that the HTTP/2-based client is able to start the playout of the video faster and thus reach synchronization faster.

V. Conclusion

In this paper, we propose a Web-based framework which enables HAS clients to synchronize their playback. Furthermore, we present a novel hybrid approach for adaptive streaming to allow fast synchronization, which relies on HTTP/2's server push in combination with super-short video segments. With the HTTP/2 approach, we are able to lower the synchronization time up to 19.4% when bandwidth is limited to 2.5 Mb/s and an RTT of 150 ms, with a reduced start-up time of 21.3%.

VI. Acknowledgments

The research was performed partially within the ICON SHIFT-TV project (under grant agreement no. 140684). Jeroen van der Hooft is funded by grant of the Agency for Innovation by Science and Technology in Flanders (IWT).

REFERENCES

[1] Cisco Systems, "Cisco Visual Networking Index: Forecast and Methodology, 2015-2020," 2016. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[2] D. Geerts, I. Vaishnavi, R. Mekuria, O. Van Deventer, and P. Cesar, "Are We In Sync?: Synchronization Requirements for Watching Online Video Together." *Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems*, pp. 311–314, 2011.

[3] M. Montagud, "Inter-Destination Multimedia Synchronization: Schemes, Use Cases and Standardization," *Multimedia Systems*, no. November 2012, 2012.

[4] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hossfeld, and P. Tran-gia, "A Survey on Quality of Experience of HTTP Adaptive Streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 469–492, 2014.

[5] V. Jung, S. Pham, and S. Kaiser, "A Web-Based Media Synchronization Framework for MPEG-DASH," *2014 IEEE International Conference on Multimedia and Expo Workshops, ICMEW 2014*, pp. 1–2, 2014.

[6] B. Rainer and C. Timmerer, "Self-Organized Inter-Destination Multimedia Synchronization For Adaptive Media Streaming," *Proceedings of the ACM International Conference on Multimedia - MM '14*, pp. 327–336, 2014.

[7] M. Kalman, E. Steinbach, and B. Girod, "Adaptive Media Playout for Low-Delay Video Streaming over Error-Prone Channels," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 6, pp. 841–851, 2004.

[8] Y. F. Su, Y. H. Yang, M. T. Lu, and H. H. Chen, "Smooth Control of Adaptive Media Playout for Video Streaming," *IEEE Transactions on Multimedia*, vol. 11, no. 7, pp. 1331–1339, 2009.

[9] M. Montagud and F. Boronat, "On The Use of Adaptive Media Playout for Inter-Destination Synchronization," *IEEE Communications Letters*, vol. 15, no. 8, pp. 863–865, 2011.

[10] B. Rainer and C. Timmerer, "Adaptive Media Playout for Inter-Destination Media Synchronization," in *2013 Fifth International Workshop on Quality of Multimedia Experience (QoMEX)*, 2013, pp. 44–45.

[11] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540, 11 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7540.txt

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

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

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

[15] Y. Fablet, E. Nassor, J. Taquet, R. De, and T. Lambert, "DASH Fast Start Using HTTP/2," in *NOSSDAV'15*, 2015, pp. 25–30.

[16] R. Huysegems, J. van der Hooft, T. Bostoen, P. R. Alface, S. Petrangeli, T. Wauters, and F. De Turck, "HTTP/2-Based Methods to Improve the Live Experience of Adaptive Streaming," *MM '15 Proceedings of the 23rd ACM international conference on Multimedia*, pp. 541–550, 2015.

[17] J. van der Hooft, S. Petrangeli, N. Bouten, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck, "An HTTP/2 Push-Based Approach for SVC Adaptive Streaming," in *NOMS 2016*, 2016, pp. 104–111.

[18] D. Yun, "Dynamic Segment Duration Control for Live Streaming Over HTTP," in *International Conference on Information Networking (ICOIN)*, Kota Kinabalu, 2016, pp. 206–210.

[19] I. M. Arntzen and N. T. Borch, "Data-Independent Sequencing with the Timing Object," in *MMSys16*, Klagenfurt, Austria, 2016.

[20] I. M. Arntzen, N. T. Borch, and C. P. Needham, "The Media State Vector," *Proceedings of the 5th Workshop on Mobile Video - MoVid '13*, p. 61, 2013.

[21] S. Wei and V. Swaminathan, "Low Latency Live Video Streaming over HTTP 2.0," *NOSSDAV'14*, pp. 1–5, 2014.