IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 27, 2018, accepted July 24, 2018, date of publication August 3, 2018, date of current version August 28, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2863033

Performance Characterization of Low-Latency Adaptive Streaming From Video Portals

JEROEN VAN DER HOOFT, (Student Member, IEEE), CEDRIC DE BOOM, STEFANO PETRANGELI, (Student Member, IEEE), TIM WAUTERS, (Member, IEEE), AND FILIP DE TURCK, (Senior Member, IEEE)

IDLab, Department of Information Technology, Ghent University-imec, 9000 Ghent, Belgium

Corresponding author: Jeroen van der Hooft (jeroen.vanderhooft@ugent.be)

The work of J. van der Hooft was supported by the Grant of the Agency for Innovation by Science and Technology in Flanders (VLAIO). The work of C. De Boom was supported by the Grant of the Research Foundation-Flanders (FWO). This work was supported in part by the imec PRO-FLOW under Project 150223 and in part by the "Optimized source coding for multiple terminals in self-organising networks" under Project G025615N.

ABSTRACT News-based websites and portals provide significant amounts of multimedia content to accompany news stories and articles. In this context, the HTTP adaptive streaming is generally used to deliver video over the best-effort Internet, allowing smooth video playback and an acceptable Quality Of Experience (QoE). To stimulate the user engagement with the provided content, such as browsing between videos, reducing the videos' startup time has become more and more important: while the current median load time is in the order of seconds, research has shown that the user waiting times must remain below two seconds to achieve an acceptable QoE. In this paper, four complementary components are optimized and integrated into a comprehensive framework for low-latency delivery of news-related video content: 1) server-side encoding with short video segments; 2) HTTP/2 server push at the application layer; 3) server-side user profiling to identify relevant content for a given user; and 4) client-side storage to hold proactively delivered content. Using a large data set of a major Belgian news provider, containing millions of text- and video-based article requests, we show that the proposed framework reduces the videos' startup time in different mobile network scenarios by over 50%, thereby improving the user interaction and skimming available content.

INDEX TERMS HTTP adaptive streaming, HTTP/2 server push, H.264/AVC, Quality of Experience, user profiling.

I. INTRODUCTION

In recent years, news providers have started to produce significant amounts of multimedia content to accompany news stories and articles. News providers such as the New York Times[1] and the Washington Post[2] now provide a large number of video-based news articles, containing individual topics or full news broadcasts. To encourage consumers to use the provided services, facile user interaction while browsing new content and skimming videos is of the utmost importance. In this context, reducing the videos' startup time has become more and more important: while videos generally take in the order of seconds to load, research has shown that user waiting times must remain below two seconds to achieve acceptable Quality of Experience (QoE) [1].

Nowadays, news content providers generally use HTTP Adaptive Streaming (HAS) to deliver video content over the best-effort Internet. In HAS, video is encoded at different quality levels and temporally divided into multiple segments with a typical length of 2 to 30 seconds [2]. As illustrated in Figure 1, an HAS client requests these video segments at the most appropriate quality level, based on e.g., the available bandwidth and the amount of buffered content. To this end, a client-based heuristic is used which attempts to optimize the 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 [3]. The client stores the incoming segments in a buffer, before decoding the sequence in linear order and playing out the video on the user's device.

This approach generally enables smooth video playout, and therefore results in a higher QoE than traditional video streaming techniques. Because of this, major players such as

[1]https://www.nytimes.com/
[2]https://www.washingtonpost.com/

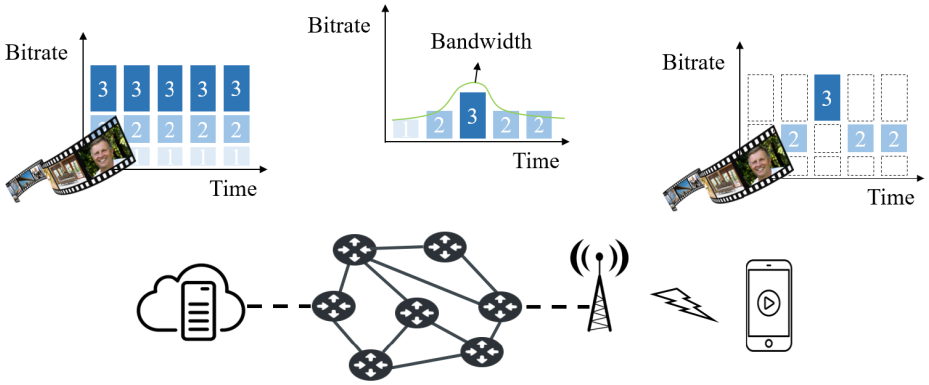

FIGURE 1. The concept of HAS. At server-side, the video is temporally segmented and encoded at different quality levels. The client requests the video segments at the most appropriate quality level, and plays them out in linear order.

Microsoft, Apple and Adobe 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 HAS [4].

HAS is well-suited for video-on-demand (VoD) scenarios, and is therefore put to good use by content providers such as Netflix[3] and YouTube.[4] The startup time in these types of scenarios is however in the order of seconds, with variations depending on the type of network connection. One of the reasons for this is that a significant number of resources need to be delivered before the video can start to play: the web page, the video player, the video's media presentation description (MPD) file, the video's (optional) initialization segment and the different video segments. Especially in mobile networks, where the available bandwidth is limited and the network latency is relatively high, this will have a significant impact on the video's startup time. A second reason is found in the segment duration, which is typically in the order of one to ten seconds: longer segments simply take longer to deliver, and thus result in higher video startup times.

Research has shown that reducing the startup delay in HAS is relevant, although it cannot occur at the cost of playout freezes or a reduced video quality: as users are used to some delay before the start of the playback, they usually tolerate it if they intend to watch the video [5]. However, when browsing through videos, i.e., when users start a larger number of videos but only watch parts of it, initial delays should be low for optimal acceptance [6]. To address this specific use case, a framework is presented for low-latency delivery of news-related HAS content, in a VoD scenario. This framework integrates four complementary optimizations in the content delivery chain:

1) Server-side encoding, to provide shorter video segments during the video's startup phase;
2) Changing the application layer protocol, using HTTP/2's server push to deliver resources back-to-back;

3a) Server-side user profiling to identify relevant content for each user;

3b) Client-side storage to hold proactively delivered content.

Each of these optimizations can be used separately, although they are very complementary. HTTP/2 server push, for example, can be used to deliver short video segments back-to-back, eliminating the need for individual requests for each of the segments. As such, content delivery and buffer rampup can happen more quickly, thus reducing the video's startup time. Prefetching news content can be done based on article recency (i.e., prefetch the n newest articles only), but also based on user profiling (i.e., prefetch based on determined user preferences), measured on an entirely different timescale. Having the right content available allows to start the video locally, thus eliminating the time needed to deliver the content over the best-effort network.

Preliminary evaluations showed that the proposed framework is able to significantly reduce the video startup time, albeit at the cost of limited network overhead and additional complexity at server- and client-side [7]. In this paper, we elaborate on each of the proposed optimizations in detail, and present a large-scale and in-depth evaluation (i.e., thousands of users, videos and video streaming sessions) on a dataset of deredactie.be,[5] an important Belgian news provider.

The remainder of this paper is structured as follows. In Section II, related work on low-latency video content delivery is discussed. The proposed framework is presented in Section III, elaborating on the advantages of each of the optimizations. The experimental setup and results are presented in Section IV, before coming to final conclusions in Section V.

[3]https://netflix.com/
[4]https://youtube.com/
[5]http://deredactie.be/

II. RELATED WORK

A large number of techniques have been proposed in literature to improve the QoE of video streaming services. These techniques can be divided in multiple ways (e.g., in client-based, server-based and network-based solutions [2]). While a lot of research has recently be done on client-side rate adaptation (e.g., BOLA and Pensieve [8], [9]), related work below mainly focuses on relevant research on low-latency end-to-end delivery on the one hand, and client-side prefetching on the other.

A. LOW-LATENCY END-TO-END DELIVERY

Since a certain amount of data must be transferred before decoding and playback can begin, startup delay is always present in HAS [2]. The minimal achievable initial delay strongly depends on the available transmission data rate and the encoder settings. Depending on the use case, the client can start playout as soon as content is available, or wait until a minimum amount of content is present. The advantage of the latter is that the buffer filling is higher when playout starts, therefore reducing the risk of buffer starvation. However, since most players start the stream at the lowest quality level, rebuffering events in the early stages of the video stream are less likely to occur. The DASH.js reference player uses a stalling threshold of 0.5 seconds by default, which is generally low enough to start playout as soon as the first segment arrives.

Quite often, the initial delay and rebuffering time are trade-off factors: reducing the startup delay at the expense of content buffering, may result in playback freezes. Hossfeld *et al*. [5] showed that in a VoD scenario for YouTube, most users tolerate an initial delay in the order of seconds, if they inted to watch the video . When browsing through videos, however, initial delays should be low for optimal acceptance [6]. Especially in the case of volatile and user-generated content, the startup delay should be low in order to maximize user engagement and acceptance. Although the focus in our evaluations is primarily on the observed video startup time, final results will also be reported in terms of buffer starvation.

A straightforward approach to reduce the startup time in HAS, is limiting the amount of data needed to start video playout. As such, the adopted encoding scheme can play an important role in the QoE of video streaming services. One possibility is to adopt the principle of scalable video coding (SVC) in HAS. This reduces the encoding and storage overhead, since each quality representation is constructed as an enhancement of the lowest quality level [10]. Because an SVC-based client has an increased number of decision points, it can cope better with highly variable bandwidth. Although SVC reduces the footprint for storage, caching and transport compared to a complete simulcast H.264/AVC system, it does introduce an encoding overhead of about 10% per layer [11]. Furthermore, since the client initially has no knowledge of the available bandwidth, most players start playout at the lowest quality level; in this case, the adoption of SVC does not impact the startup time. More recently, a number of stand-alone HAS players have adopted H.265/HEVC, a video compression standard was developed to provide twice the compression efficiency of the previous standard, H.264/AVC [12]. 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. Although its application is increasing, most browsers offer no support for HEVC at the time of writing [13]. It is worth noting that a scalable version of the standard, SHEVC, exists as well.

In live streaming scenarios, advanced client-side rate adaptation heuristics can be used in order to achieve an acceptable QoE when the buffer size is small. Recently, Shuai *et al*. [14] proposed a heuristic which allows the client to stream video with stable buffer filling. In the model for the QoE, rebuffering events - including the one at startup - are penalized by applying a suitable weight factor. In this way, high startup times and stalling events are actively avoided. In the evaluation of the proposed approach, results are however only shown in terms of the estimated QoE; as such, it is unclear how the video startup time is affected. Miller *et al*. [15] propose LOLYPOP, a rate adaptation heuristic for low-latency prediction-based adaptation designed to operate with a transport latency of a few seconds. In their evaluation, a video streaming scenario is considered in which the total delay (i.e., segment duration plus upper bound on transport latency) equals merely 5 seconds. To achieve such a low value, LOLYPOP leverages TPC throughput predictions on multiple time scales, from 1 to 10 seconds, along with estimations of the relative prediction error distributions. Using this approach, the authors are able to improve the mean video quality by a factor of 3 compared to FESTIVE, a well-known rate adaptation heuristic to improve fairness and stability in HAS [16]. Results for the video startup time are however not considered in this paper.

Although real-time streaming protocols are not applicable in the targeted use case, we can use different protocols at the application layer. In this context, the new HTTP/2 standard was published as an IETF RFC in February 2015, mainly focusing on the reduction of latency in web delivery. Since then, research has shown the application of HTTP/2 can either reduce or increase the load time [17]–[19]. In the context of video content delivery, however, significant improvements can be achieved. Wei and Swaminathan [20] first explore how HTTP/2's features can be used to improve HAS. By reducing the segment duration from 5 to 1 second, they manage to reduce the camera-to-display delay with about 10 seconds. An increased number of GET requests is avoided by pushing k segments after each request, using HTTP/2's server push. 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 [21]. The authors also 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 full-push feature, in which segments are pushed from server to client until the client specifies otherwise, or the connection is terminated. In contrast to these works, we propose to combine HTTP/2's server push with a hybrid segment duration scheme, resulting in smoother buffering and faster startup.

Optimizations can also be performed on the transport layer. In recent work, Gatimu *et al*. [22] showed that the Flexible Dual TCP-UDP Streaming Protocol (FDSP), which combines the reliability of TCP with the low-latency characteristics of UDP, can be used to reduce the video startup time. In the considered setup, FDSP delivers the more critical parts of the video data via TCP and the rest via UDP. The authors show that this approach results in significantly less rebuffering than TCP-based streaming and a lower packet loss rate than UDP-based streaming. Although results are promising, the evaluated approach does not map to traditional HAS, in which segments are retrieved back-to-back and have to be fully downloaded before playout can start.

When it comes to encoding, new real-time technologies such as Web Real-Time Communication (WebRTC) have recently been introduced [23]. Where HAS is generally used in VoD scenarios or live streaming scenarios where the delay can be in the order of tens of seconds, such as sports events, these technologies focus on collaborative real-time video streaming communication where the delay should be in the order of a few hundreds of milliseconds. Furthermore, they have been developed with a peer-to-peer architecture in mind, where a small group of clients can directly communicate with each other. Since each sender needs to encode a separate stream for each of the receivers, this approach suffers from scalability issues when many participants are present at the same time. Although useful in real-time communication, these technologies do not envision traditional VoD scenarios and are thus unsuitable to provide the required low-latency aspects in HAS.

Recently, further improvements to HAS have been made, such as Server and Network Assisted DASH (SAND) [24]. While the focus of this paper is on over-the-top video delivery only (i.e., delivery without control over the network), SAND aims to further improve performance by enabling in-network decisions. 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. This allows to trigger control mechanisms such as flow prioritization, bandwidth reservation and video quality adaptation based on the network's and client's current state. A large number of studies has been conducted, showing that the SAND principle can significantly improve the QoE in HAS [25], [26]. This concept however moves away from over-the-top solutions, and is therefore not considered in this paper.

B. PREFETCHING OF MULTIMEDIA CONTENT

To improve the QoE in video streaming, content can be brought closer to the end user. Streaming providers have massively adopted the use of Content Delivery Networks (CDN), reducing the load on the origin server and serve the video with lower latency and increased bandwidth. General caching strategies can be applied, taking into account characteristics such as content popularity, user preferences, the client's location etc. In this regard, a number of studies have shown significant improvements. As an example, Krishnappa *et al*. [27] exploit particular user behavior to improve the caching efficiency of YouTube videos. By rearranging the related video list to give preference to cached videos, the cache hit rate is improved by a factor of 5. Caching strategies can also be improved when future user requests are known in advance. For example, binge-watching has become a well-known phenomenon for video streaming services, meaning that users tend to watch multiple episodes of the same TV show consecutively. As shown by Claeys *et al*. [28], this information can be used to estimate future video segment requests and improve caching efficiency. Apart from being cached in a CDN, content can also be prefetched by the client. Krishnamoorthi *et al*. [29], for instance, propose a scheme in which three policy classes are applied to preload content for HAS. The authors however assume that the content provider has already established a list of relevant content which should be prefetched. In this work, we also focus on how to select this content at server-side.

For this purpose, we will use techniques that have been proposed in the context of recommender systems, where users and their consumed items are projected in a low-dimensional vector space [30]. Users with similar consumer characteristics will typically have vector representations that lie close to each other in this space, while dissimilar user vectors are located far apart. Matrix factorization, for instance, is an often used technique to arrive at a vector space with such characteristics. It is typically applied to a user-item rating matrix, while imposing that the scalar product between a user and item vector is a good rating predictor. The problem with this approach, however, is that the user vectors are considered static, which is not always ideal in dynamic scenarios in which many items are consumed one after the other, such as songs, videos and news content. It has recently been shown in literature, and real-life scenarios at Netflix and Spotify, that it is often beneficial to explicitly consider the time aspect by modeling users in a dynamic fashion [31]–[33]. In these systems, a user is typically represented by aggregating item vectors across time. For example, in the work by Hidasi *et al*. [34] a recurrent neural network is used to process items one after the other; the output of this neural network is a set of recommended items after processing a new item. The downside of this approach is that users are not modeled explicitly in the same space as the items, and therefore does not allow for direct user-item comparisons. In order to achieve this, we can simply sum or average the

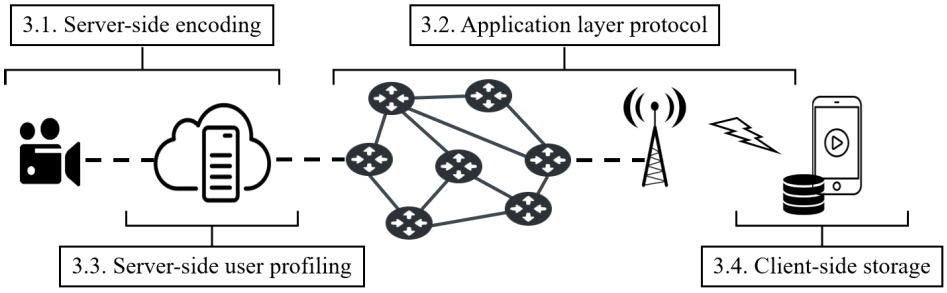

FIGURE 2. The proposed HAS delivery framework for media-rich content from news providers [7].

consumed item vectors, through which we remain in the same item space. We will further elaborate upon this in the next section.

III. PROPOSED FRAMEWORK

The proposed framework integrates four complementary optimizations in the content delivery chain, as illustrated in Figure 2. First, we consider the aspect of video encoding, using a shorter video segment duration to reduce the playout delay. This optimization requires sufficient resources at server-side, in order to encode provided content both for multiple quality representations and for different segment durations. Second, we focus on the applied application layer protocol, discussing the possibilities of HTTP/2's server push feature. This requires an HTTP/2-enabled server, equipped with a custom request handle to push required resources from server to client. Since most browsers nowadays have full support for HTTP/2, no changes to the client are required. Third, we consider user profiling as a way to predict user interest and interaction. To this end, the server needs to monitor all incoming requests and keep track of several content- and user-based characteristics. Fourth, client-side storage is considered to store content which is proactively delivered to the user, once it is deemed of interest by the profiling component. This requires additional complexity at client-side, and is prone to bandwidth overhead when the wrong content is prefetched. Below, we elaborate on each of these optimizations in detail.

A. SERVER-SIDE ENCODING USING HYBRID SEGMENT DURATION

The first part of the proposed framework consists of server-side encoding, and more specifically on the segment duration of the provided content. As found in previous work, reducing the duration of video segments comes with a number of advantages [35]. Most importantly, the short segments require a lower download time, resulting in a reduced delivery time and thus in faster startup. However, 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 compared to segments of higher length. This encoding overhead was analyzed for seven videos at multiple frames per seconds (FPS) in previous work, showing that a segment duration of 1 second results in a bitrate overhead between 12.1 and 22.4% compared to a segment duration of 8 seconds, and in an overhead between 28.7 and 49.9% for a segment duration of 250 ms [35]. Moreover, since a unique request is required to retrieve each 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 is in the order of 100 ms, depending on the network carrier and the type of connection.

While traditional streaming solutions use a fixed segment duration in the order of 2 to 30 seconds, we propose to use different segment durations for the startup and steady-state phase of the video streaming session. This allows us to both reduce the video startup time by using short video segments in the startup phase, and overcome the aforementioned issues by switching to longer segment durations once the video is steadily playing. Two approaches are possible: (i) initially start at the lowest segment duration d_1, switching to the highest segment duration d_n once a significant amount of segments has been downloaded, and (ii) initially start at the lowest segment duration d_1, switching to d_2, d_3, ... until a segment duration of d_n is reached. The advantage of the latter is that the buffer is ramped up smoothly, preventing possible freezes when switching from the lowest to the highest segment duration when the buffer level is relatively low. A disadvantage to this approach is that multiple versions of the content need to be available, each containing a different segment duration. However, since the segment duration is changed only during the startup phase, only the first part of the content needs to be encoded multiple times. Furthermore, since most players generally start playout at the lowest video bitrate, it is sufficient to provide multiple segment durations for the lowest quality representation only.

There are multiple ways to apply the proposed segment duration scheme. One possible approach is to generate the MPD, which contains relevant information on the available content (e.g., the video's duration and available quality representations), in such way that several parts of the video are distinguished [36]. It is then possible to define different segment durations for different quality representations and time intervals, making the approach completely DASH-compliant. A second approach is to limit the segment duration in the

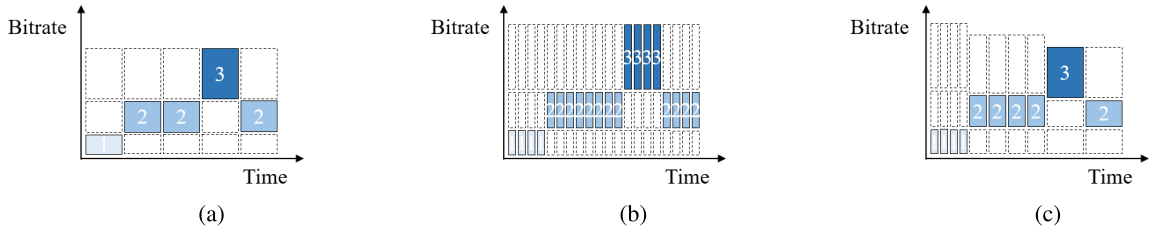

FIGURE 3. Possible segment duration schemes in HAS. While traditional schemes use a fixed segment duration, the proposed hybrid scheme changes the segment duration over time, smoothly ramping up the video player's buffer. (a) Long segment duration scheme. (b) Short segment duration scheme. (c) Hybrid segment duration scheme.

video player itself, for instance by tracking the video playout progress and available quality representations.

It is worth noting that the proposed scheme can not only be applied at startup, but each time the buffer filling drops below a certain threshold: the player can recover more quickly, and thus reduce the total time of stalling. In this case, however, the full video needs to be available in different segment durations. This scheme is not adopted in this paper.

B. APPLICATION LAYER OPTIMIZATIONS USING HTTP/2'S SERVER PUSH

At the start of an HAS video streaming session, a large number of files need to be downloaded. In a stand-alone client, a request is first sent for the video's MPD file. Based on the contents of this file, the client proceeds to download the initialization segment (if any) and from then on, requests video segments one by one. In a web-based context, the HTML page and its required resources need to be fetched as well, including the HAS player, JavaScript sources, CSS files, images, etc. All these resources are requested over HTTP, which among others, allows to traverse firewall and NAT devices, and reuse the existing delivery infrastructure. Retrieving these resources one by one takes time, given RTT is lost for every request. Especially in mobile networks, where the latency is in the order of tens to hundreds of milliseconds, this can have a significant impact on the startup time.

When it comes to browser page loading, the total time can be reduced by using up to six parallel HTTP/1.1 connections. This allows to retrieve resources faster, since idle RTTs can be omitted. It is however infeasible to use this approach in the case of HAS: during the startup phase, the client's browser does not know in advance which resources will be required next (e.g., the download of the DASH.js player cannot start before the HTML source code is parsed), and once the video session has started, segments are retrieved one by one to guarantee in-order delivery, and thus the (timely) arrival of the most crucial segments first. An alternative way to deliver required resources more quickly, would be to forward these without the client actually requesting them. To this end, HTTP/2's server push can be used.

In Figure 4a, the example scenario from Figure 3c is illustrated for HTTP/1.1. Here, the HTML source code is downloaded first, and is parsed by the browser. This code includes the DASH.js script, which is in its turn requested. Once the player is present, it initiates the download of the video session's MPD and parses the document. It finds out that an initialization segment is required for each of the quality representations, and issues a request for the one corresponding to the lowest quality. From then on, segments are requested one by one, evaluating the available bandwidth once each of the segments has been downloaded.

In 2015, the HTTP/2 standard was published as an IETF RFC. Its main purpose is to reduce the latency in web delivery, using request/response multiplexing, stream prioritization and server push. The latter can be used to push video segments from server to client, without the client sending a GET request for the required resources. Pushing the content back-to-back allows to eliminate idle RTT cycles, reducing buffering time and improving bandwidth utilization.

In previous work, a stand-alone client was considered to evaluate the use of HTTP/2's server push feature [35]. In this context, we propose to push resources related to the video streaming session only: once the manifest is requested, video segments are continuously pushed to the client, until an explicit stop request is sent or the connection with the client is terminated. In this way, idle RTTs during content download are eliminated, resulting in reduced delivery times and thus faster video startup.

In this work, we consider the browser-based DASH.js reference player, with the hybrid segment duration scheme presented above. As illustrated in Figure 4b, HTTP/2 server push is now used to deliver the HTML source code of a sample web page, the DASH.js reference player embedded within this page, the MPD, the initialization segment and the first k video segments, corresponding to the first x seconds of the the video stream. From then on, one GET request can be used to retrieve each x seconds of video. Depending on the selected segment duration, the server can push a different amount of segments, while the client can specify the desired quality representation based on its rate adaptation heuristic. Similarly, this approach results in the removal of idle RTTs, reducing the video's startup time and increasing the total throughput.

Although not applicable in the evaluation setup in Section IV, it is worth noting that additional sources, such as images, scripts and CSS, can be retrieved using HTTP/1.1

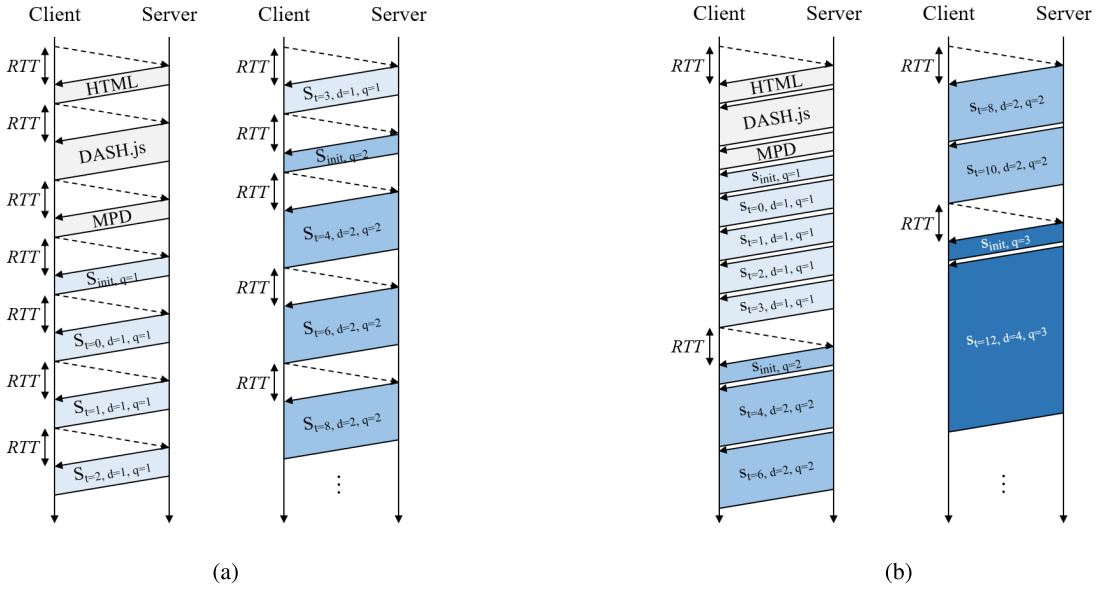

FIGURE 4. Sequence diagrams of the proposed hybrid segment duration schemes. Using HTTP/1.1, each video segment requires a separate GET request. Using HTTP/2, one request allows to deliver multiple resources, thereby reducing the video's buffering and startup time. (a) HAS over HTTP/1.1. (b) HAS over HTTP/2.

(potentially with multiple connections) or by HTTP/2 (potentially with server push), or from the browser's cache if present.

C. SERVER-SIDE USER AND CONTENT PROFILING

A third optimization consists of server-side user and content profiling. Its purpose is to build a profile for all platform users, determining their preferences towards certain news content. Generally speaking, the purpose of the profiling component is to select a subset of relevant, recent video content for any given user. This content can then be prefetched by the client, effectively reducing the video startup delay at the time of request. In the proposed framework, we thus want a means to determine such a subset. To this end, for each of the users, we compare the performance of each of the following recommendation strategies over a recent period (i.e., the last n number of requests):

- **Likely to consume most popular content** - Certain video articles have a higher probability of being requested than others. Research has shown that this depends, among others, on the type of subject, the location of the event and the objectivity of the title [C]. Popular articles are generally highlighted on (top of) the home page, and are more often shared on social networks such as Facebook and Twitter. If we want to consider a relevant subset of videos to prefetch, it thus would make sense to consider the most popular content only. When this profile is assigned, the number of requests issued within a certain time interval is considered to rank the available content. From this list, the n most popular articles are considered of interest to the user;
- **Likely to consume most recent content** - News content is typically short-lived, i.e., its relevance decreases quickly over time [37]. News articles and videos are only featured on (top of) the home page for a limited amount of time only, and are quickly replaced by new data and news topics. Furthermore, a significant amount of users prefer to stay up-to-date by visiting news web sites multiple times a day, making news older than a couple of hours less relevant. Therefore, when this profile is assigned, all content is ranked according to the time of publication. From this list, the n most recent articles are considered of interest to the user;
- **Likely to consume personalized content** - Some users are interested in particular news subjects and articles, and therefore consume specific videos only. For this type of users, we will, as traditionally done in recommender systems, represent each user and video article by a low-dimensional vector. To this end, we assume that every video has associated textual metadata, and apply a natural language processing model to represent each of the articles. This metadata can include, but is not limited to, the author(s), title, summary and text-based content of the news article. Based on previous work, word2vec is selected to accomplish this [7], [38]. Word2vec learns low-dimensional word vectors, also called word embeddings, by training a two-layer neural networks to reconstruct linguisitic contexts of words in a large text corpus. For each word in this corpus, a word embedding is learned that is located in a low-dimensional space such that words with a common context are positioned close to one another [38]. Since word2vec operates on word level, we will represent each article by the sum of the word vectors it contains. A user is represented by a vector as well, which is initially an all-zeros vector. Each time a new article is requested by a

user, the corresponding vector is updated by summation of the user and article vector in an online fashion. This approach allows us to create a unique vector for each user, building a user profile over time. The relevance of an article $\vec{a}$ to a user $\vec{u}$ can then be determined using the cosine similarity:

$$\cos(\vec{u}, \vec{a}) = \frac{\vec{u} \cdot \vec{a}}{\|\vec{u}\|_2 \|\vec{a}\|_2}. \quad (1)$$

The higher this similarity, the higher the user's preference towards an article is assumed. When this profile is assigned, recent content is considered and ranked according to the cosine similarity between the user and article vectors. From this list, the n most similar articles are considered of interest to the user. Note that this approach is prone to the cold-start problem, since the user vector is initially set to zero and is slowly built up for each user.

We propose to assign each user one of the aforementioned profiles, but do it in such way that performance does not suffer from the cold-start problem and that a user's preferences can change over time. To this end, each user is initially assigned the popularity profile, where the most requested articles are considered. Once a sufficient amount of requests has been issued by the user, we allow the assigned profile to be changed throughout time, using online evaluations of the user behavior and preferences towards certain content. To determine the most appropriate category for a given user, and therefore the best approach to rank the available content, a sliding window is used over the user's past n requests. Within this window, the position of each of the user's requests is evaluated in the sorted list generated by each of the three profiles. The user is then be mapped to the profile which results in the lowest average position of the requests.

Using the average position for the last n requests as a metric, differences between the three profiles can be small. This can result in a large number of switches in the assigned profile, which should preferably be avoided. For this reason, hysteresis is applied to only change the assigned profile when the metric of a certain profile outperforms the currently assigned profile by at least a fraction α.

The result of the proposed approach is a shortlist containing the most relevant video articles for each of the users. As explained below, the content of this shortlist can be used to proactively deliver the content to the user, anticipating future article requests.

D. CLIENT-SIDE STORAGE

A final component of the proposed framework consists of client-side storage, which is used to enable proactive delivery of relevant video content. If the right content is sent, using such approach allows to significantly reduce the video session's startup time. Depending on the use case scenario, multiple options for content delivery and client-side storage are possible. In a stand-alone application, a dedicated cache on the local device can be used. Based on server recommendations, the application can retrieve content in the background. Measure needs to be taken as not to use prefetching too aggressively, as it introduces a risk of increasing battery drain and bandwidth use. In web-based applications, control over client-side storage is less evident. Recent versions of browsers such as Google Chrome allow to prefetch web pages which are referred to in the current page. Pages are prerendered in a hidden tab, and moved to the foreground upon request. This can be extended as to provide support for HAS, retrieving the first x seconds of embedded video content. Most browsers now also support HTTP/2, storing pushed resources in the browser's cache. This allows the server to push additional resources upon a client's incoming request, yet care needs to be taken that prefetching does not interfere with the transfer of more urgent resources (e.g., the current video being played out). For this reason, the best approach is to only deliver additional resources once the requested article page has been retrieved completely.

Regardless of how the content is delivered, it is important that the right content is sent. Indeed, proactive delivery of non-relevant content results in network overhead, since bandwidth is wasted on content which may never be consumed. Therefore, we can apply the profiling detailed in Section III-C to determine a shortlist of articles, and consider the top k articles for prefetching. In our evaluations in Section IV, we investigate the impact of this parameter (which corresponds to the *cache size*) on the startup time (no requests need to be sent to the server in case of a cache hit) and the network overhead. Note that only the first d_n seconds of video, corresponding to the largest segment duration, are stored: as soon as video playout has started, the remaining content is fetched from the server as in a traditional streaming scenario.

IV. EVALUATION

In this section, we evaluate the impact of the proposed framework on the video startup time under realistic network conditions. We first discuss the considered use case of a major Belgian news provider, and present the experimental setup used to evaluate results. We then present the most important results, discussing the advantages and shortcomings of each of the proposed optimizations.

A. USE CASE: DEREDACTIE.BE

Deredactie.be is one of the major news websites in Belgium, hosted by the Flemish Radio and Television Broadcasting Organization (VRT). In recent years, its focus has shifted largely from simple text-based articles towards multimedia-rich news reports. Because of this, the website is an excellent use case for the proposed delivery framework. On the home page, users are presented an overview of recently published content, containing a relevant poster, a title and a short introduction on each of the news topics (Figure 5). In the ''video zone'', a separate page, all videos published in the last week are presented in order of publication (i.e., most recent first).

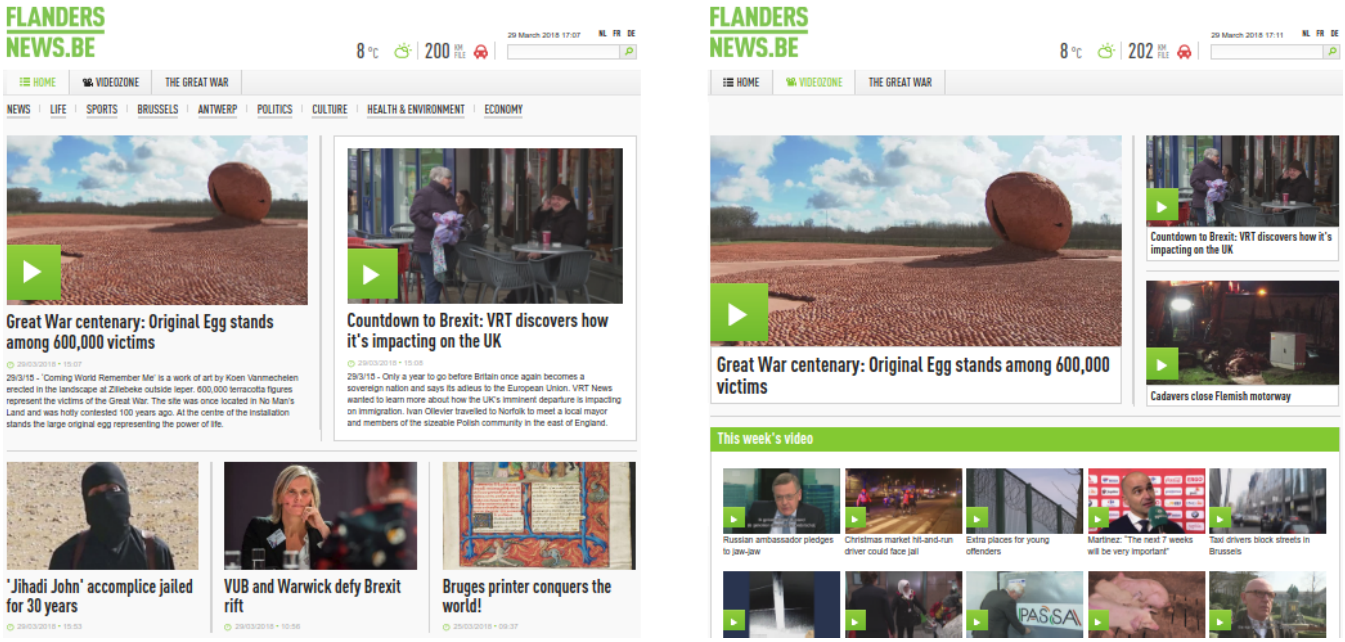

FIGURE 5. Considered use case of deredactie.be, homepage (left) and video zone (right).

The website also contains references to other (news) sources, such as Sporza (sports) and Canvas (culture).

In collaboration with VRT, Van Canneyt *et al*. were able to collect a data set containing approximately 300 million website requests, issued between April 2015 and January 2016 [37]. For every request to the website, among others the requested URL, the referrer URL, the server's and client's local time, and the client's hashed IP and cookie ID were logged.

From the given dataset, all users and article requests were extracted. The HTML and XML sources of the requested articles were retrieved, and relevant information such as the title, summary and content was extracted. All embedded video was retrieved as well, resulting in a total of 19,437 videos. All data was used by the proposed framework, as detailed below.

B. EXPERIMENTAL SETUP

In the remainder of this section, we evaluate the proposed framework under realistic network conditions. To this end, a network setup is emulated using MiniNet, where a client is connected to an HTTP/1.1- and HTTP/2-enabled Jetty server (Figure 6). As for the mobile network scenario, both 3G and 4G network scenarios are considered; while 4G coverage in Belgium is excellent, clients are still often forced onto 3G in areas with a lower population density [39]. To emulate network conditions, traffic control is used to set the network latency to 120 and 60 ms for 3G and 4G respectively, and to shape the available bandwidth of the client according to bandwidth traces provided by Riiser *et al*. and van der Hooft *et al*. [40], [41]. The client uses the Google Chrome browser in headless mode to start a video streaming session, using the reference DASH.js player.

FIGURE 6. Experimental setup. MiniNet is used to host a virtual network within a Docker container. The DASH.js player is used in the Google chrome browser, starting and playing different video streaming sessions from the HTTP/2-enabled Jetty server.

The open-source code of the Jetty server is slightly modified, allowing it to push the required HTML and JavaScript resources, the MPD, the initialization segment and the first ten seconds of a given video upon request. To allow seamless connection over HTTP/2, a Node.js proxy is provided for each client. This proxy can retrieve content from the local filesystem as well, and can therefore be used to measure startup times for prefetched video. The complete setup has been wrapped in a docker container, increasing portability and allowing parallel execution of video streaming sessions. Experiments were carried out on five physical nodes on the Virtual Wall,[6] with twelve docker containers running simultaneously on a hexa-core Intel(R) Xeon(R) CPU E5645 @ 2.40GHz with 24 GB of RAM. Below, results are shown for different scenarios, including an additional optimization in each experiment.

C. SHORT SEGMENT DURATION

Since shorter video segments require less data to be transferred from server to client, the client should be able to

[6]http://doc.ilabt.imec.be/ilabt-documentation/

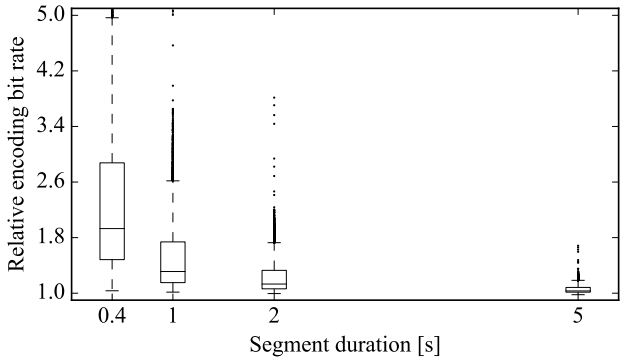

FIGURE 7. Encoding bitrate as a function of the segment duration, relative to the bitrate for a segment duration of 10 seconds. Outliers for a segment duration of 0.4 seconds go as high as 14.065, but are omitted in favor of readability.

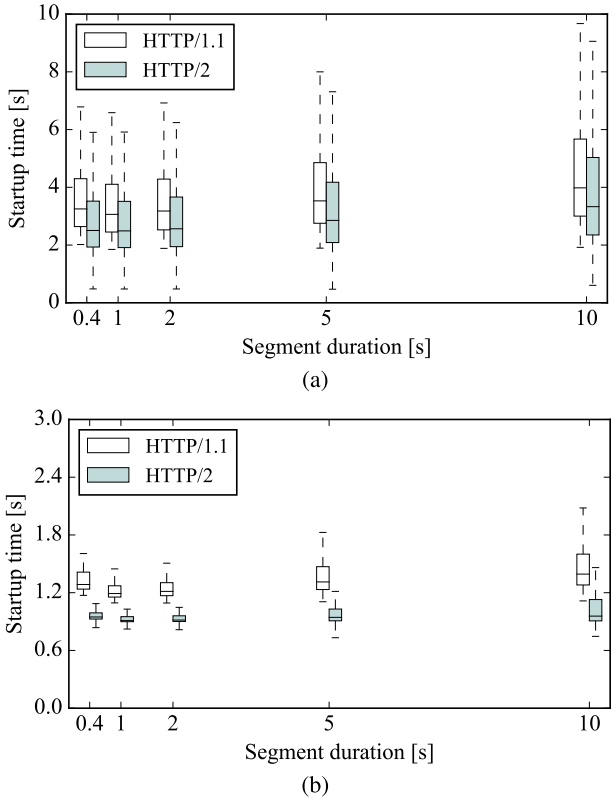

FIGURE 8. Startup time as a function of the segment duration. Outliers, corresponding to at most 7% of the data, have been omitted in favor of readability. (a) 3G network. (b) 4G network.

start playout faster. As mentioned above, this approach introduces an encoding overhead. In previous work, we evaluated this overhead for shorter video segments on seven different videos [41]. Given the extent of the presented dataset, we decided to evaluate the encoding overhead for all 19,437 video articles published within the time of logging. By default, deredactie.be provides its video content at a frame rate of 25 FPS, a spatial resolution of 640 × 360 and a segment duration of ten seconds. This content was re-encoded using AVC/H.264 with the same frame rate and resolution, but with a segment duration ranging from 400 ms to 10 seconds. 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 values ranging from 10 to 250 respectively. To realize the same visual quality, the Constant Rate Factor (CRF) rate control in the x264 encoder is enabled, with a CRF value of 25. This results in average video bit rates of 423 and 231 kb/s for a segment duration of 400 ms and 10 seconds respectively.

Figure 7 shows a boxplot of the encoding overhead for different segment durations, relative to the bit rate of a segment duration of 10 seconds. The obtained average video bit rates equal 423, 300, 261, 238 and 231 kb/s for a segment duration of 0.4, 1, 2, 5 and 10 seconds respectively, or a relative overhead of 83.3, 30.4, 13.2 and 3.3% compared to a segment duration of 10 seconds. Values for the overhead range from 1.035 (low overhead) to 14.065 (significant overhead) for a segment duration of 0.4 seconds. The former stems from videos with a lot of movement and scene switches, for which the insertion of additional IDR frames has almost no impact on the overall video bit rate (e.g., a report on the world record pillow fight), while the latter stems from videos with hardly any movement at all (e.g., still photos during a news broadcast). This shows that the encoding overhead can be significant, and should be avoided whenever possible. Therefore, applying a hybrid approach in which short segments are used at startup and long segments in steady-state, is beneficial in terms of consumed bandwidth and video quality.

In a first video streaming experiment, we evaluate the startup time for different segment durations d_1, where the required content is requested over HTTP/1.1. The segment duration ranges from 0.4 to 10 seconds, at a frame rate of 25 FPS. Figure 8 shows the boxplots for the startup time of 19,437 video streaming sessions, one for each video in the dataset. Outliers, corresponding to at most 7% of the data, have been omitted in favor of readability: some outliers reach values up to 38.4 seconds because the throughput traces contain different periods of low throughput, where connection is bad to non-existing.

In a 3G scenario, the observed gains are significant: the median startup time is reduced from 4.0 to 3.1 seconds for a segment duration of 10 and 1 seconds respectively. Furthermore, variability is significantly lower, reducing high startup times which generally impede the QoE the most: the 90% and 99% quantiles, for instance, are reduced from 8.4 to 5.9 seconds and from 15.7 to 11.3 seconds. Results also show that the segment duration cannot be reduced indefinitely: for a segment duration of 0.4 seconds, the median startup time increases again to 3.3 seconds, and the 90% and 99% quantiles to 6.1 and 11.4 seconds respectively.

In a 4G scenario, the gains are less outspoken: the median startup time is reduced from 1.4 to 1.2 seconds for a segment duration of 10 and 1 seconds respectively, while the 90% and 99% quantiles are reduced from 1.9 to 1.5 seconds and from

4.2 to 3.4 seconds. Because the throughput is signi-ficantly higher, the importance of the video file size is simply reduced. Relatively speaking, however, the observed gains are still significant: a reduction of the median startup time of 14.5% is achieved for 4G, compared to 23.1% for 3G.

D. SHORT SEGMENT DURATION AND SERVER PUSH

As explained in Section III, the application of HTTP/2 server push allows to avoid idle RTT cycles in the buffer rampup phase. In Figure 8, the startup time of all video streaming sessions is compared between HTTP/1.1 and HTTP/2. In a 3G scenario, where the latency is equal to 120 ms, the median startup time is reduced from 4.0 to 3.3 seconds (−17.5%) and from 3.1 to 2.5 seconds (−19.3%) for a segment duration of 10 and 1 seconds respectively. In a 4G scenario, where the latency is equal to 60 ms, the median startup time is reduced from 1.4 to 1.0 seconds (−28.6%) and from 1.2 to 0.9 seconds (−25.0%) for a segment duration of 10 and 1 seconds respectively. These gains are a direct consequence of back-to-back delivery of the required video resources. It is worth mentioning that the application of HTTP/2 server push has no impact on variability: the reduction of the startup time is linearly correlated to the network delay only.

E. USER AND CONTENT PROFILING

The application of user profiling is situated on a different timescale than the actual video streaming. For this reason, we first show results for the proposed user profiling strategies, before applying them in a video streaming scenario. To enable user profiling, the title, summary and text content from each article published within the time of logging was extracted, Dutch stopwords were eliminated and the resulting lower-case text was used as input to train a word2vec-based model with 100 dimensions. Note that all articles containing video, include at least a title and a brief summary; therefore, each article can be represented by the sum of its word vectors.

To evaluate the accuracy of the resulting models, we replayed all article requests issued by users who were active during at least half of the period of logging (i.e., five months) and who, on average, requested at least one video article per day. Eliminating page crawlers, this resulted in a pool of 5,835 users, who together request a total of 1,848,319 video articles. Similar to articles, each user is represented by a 100-dimensional vector, which is updated each time an article is requested.

Before each request, a sliding window over recently issued requests is updated. Within this window, each of the three strategies (i.e., popular, recency and word2vec) is evaluated based on the rank of the articles in the sorted list, and a decision is taken to either stay on the current strategy, or switch to an alternative one if this has shown to result in a better performance. We performed an analysis of different parameter settings, including the optimal window size and hysteresis fraction thereof.

A higher window size entails more information, and thus a better decision regarding which category to assign to the user. However, using a window size which is too large, users which are likely to switch to a different category remain at the (initially assigned) popular strategy too long, resulting in lower performance. After careful consideration, a window size of 100 requests was selected.

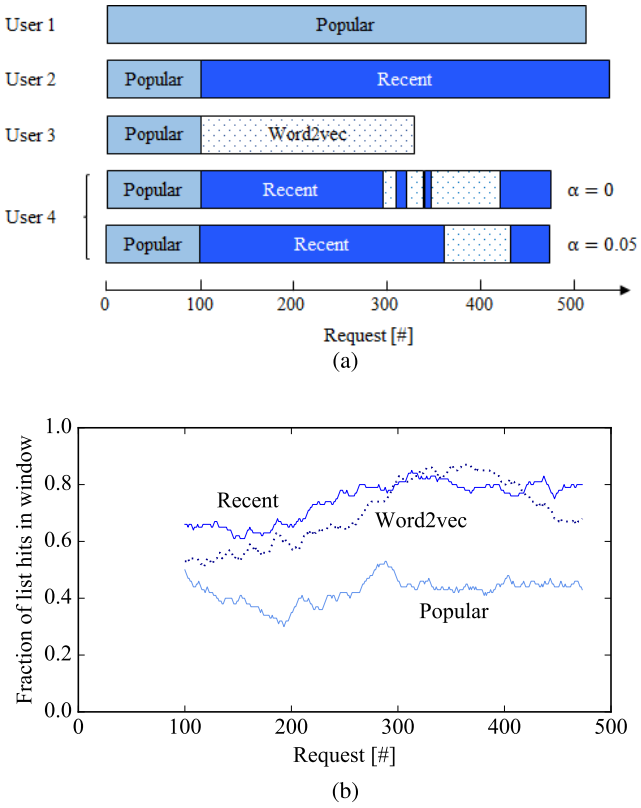

FIGURE 9. Category preference and switching for four different users (left) and the accuracy in the sliding window for user 4 (right), for a list size of 16 articles and a window size of 100 requests. Because results for the recent and word2vec profile overlap, a large number of switches can occur. (a) Switching behavior for four example users. (b) Accuracy within the sliding window for user 4.

When changing the assigned profile based on the number of list hits within the sliding window, differences in accuracy can be small. This can result in a large number of switches in the assigned recommendation strategy. To avoid this from happening, hysteresis is applied to only change the strategy when the cache hit ratio of a certain approach outperforms the currently assigned approach by at least a fraction α of the window size. As illustrated in Figures 9a and 9b for an example user, using a value of $\alpha = 0.05$ is sufficient to reduce the number of otiose strategy switches.

Figure 10 shows results for the three strategies and for the combined approach. On the x-axis the considered article list size (i.e., the number of articles that the recommendation algorithm is able to select) is presented, on the y-axis the relative number of requested articles which were present in this list. As an example, using a list containing the 16 most recent articles at each point in time, 43.3% of requested articles were present in the list. Compared to a static popularity

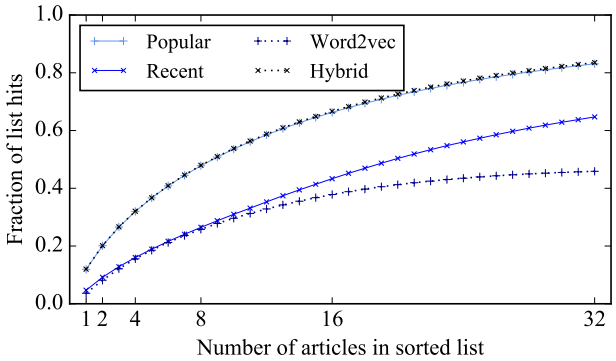

FIGURE 10. Relative number of list hits for the three different user profiles, and the proposed hybrid approach with a window size of 100 requests and $\alpha = 0.05$.

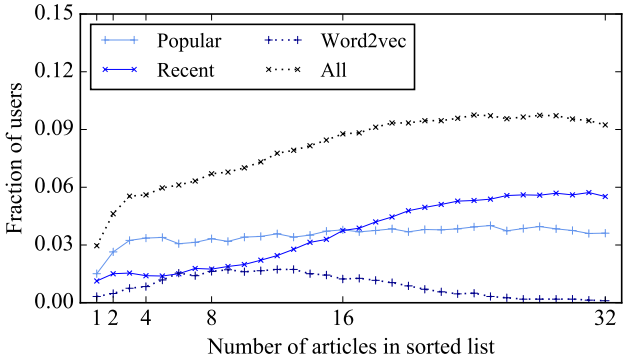

FIGURE 11. Relative number of users whose user strategy is changed over time. Users are categorized based on the dominant user strategy (either popular, recent or word2vec-based).

strategy, applying the proposed approach results in higher accuracy, although improvements are limited; as an example, the accuracy increases from 0.4781 to 0.4794 (+0.3%) and from 0.6629 to 0.6664 (+0.5%) for a list size of 8 and 16 respectively. For most users in this dataset, simply considering the most popular articles at each point in time, results in the best performance. This is because most users tend to consume content which is readily available and presented on (top of) the home page. For this reason, it is more difficult to build accurate user profiles, or predict consumption behavior based on previous requests.

Figure 11 shows the relative number of users for which the profile is changed over time (total), and the relative number of users for which either the popular, recent or word2vec-based profile is selected most often when switching. As can be observed, at most 10% of users is ever assigned a different profile than the most popular one. The recency profile is preferred by at most 6% of users, and the word2vec-based profile by 2%. As illustrated in Figure 12, however, changing the recommendation strategy for these users does result in significant improvements.

As an example, for a list size of 8, the performance relative to a static profile based on popularity, is improved by 12.5% and 11.6% for users who are most often assigned

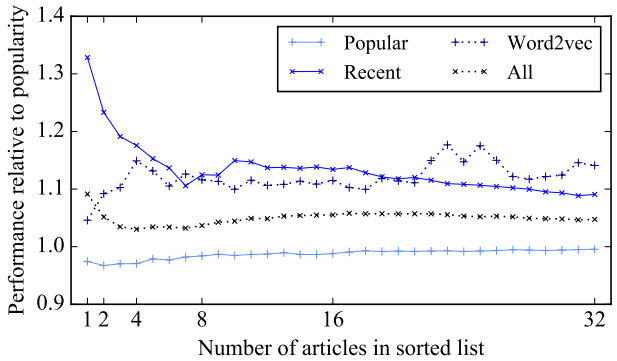

FIGURE 12. Performance for users whose recommendation strategy is changed over time, relative to a static popularity strategy. Values lower than 1 can occur, since switching is not always beneficial (although one strategy outperforms the others in the sliding window, it is not guaranteed that this will also be the case in the requests to come).

the recency and word2vec-based strategy respectively. These users have an outspoken preference towards certain topics and TV programs, which, when taken into account, in some cases improve performance by a factor of 2 and more. For these users, the proposed hybrid profiling approach can thus be adopted to improve accuracy, and therefore improve the list of content considered for prefetching when client-side storage is enabled. It is worth noting that for lower list sizes, switching the assigned profile can in some cases result in lower performance: given the low number of considered articles, there is little certainty about potential list hits.

F. SHORT SEGMENT DURATION, SERVER PUSH AND PROACTIVE PREFETCHING

In a final set of experiments, we assume that client-side prefetching and storage is also possible. To this end, the proposed recommendation scheme is adopted to rank the available content for the 5,835 users in the user pool. It is worth noting that any recommendation scheme could be used here: the proposed optimizations are complementary, and can thus be omitted or replaced by valid alternatives. Then, the full request log is replayed for each user.

In practice, the client would prefetch relevant content during off-times. In the case of a stand-alone application on a mobile device, prefetching can be done in the background. In the web browser, this can be done based on server suggestions, either when the current page has finished loading or when the playing video has been retrieved. In our experimental setup, it is however practically infeasible to recreate the full user session: only requested article URLs are available, yet not the information on the played out quality of video, the length of the session, the available throughput, etc. For this reason, prefetching is not directly implemented: we simply assume that the client has, at each point in time, stored the first $d_n = 10$ seconds of content for each of the top n video-based articles. When the client issues a request, the content is retrieved either from local storage (i.e., through the proxy) if the content is available in this list, or from the server if not.

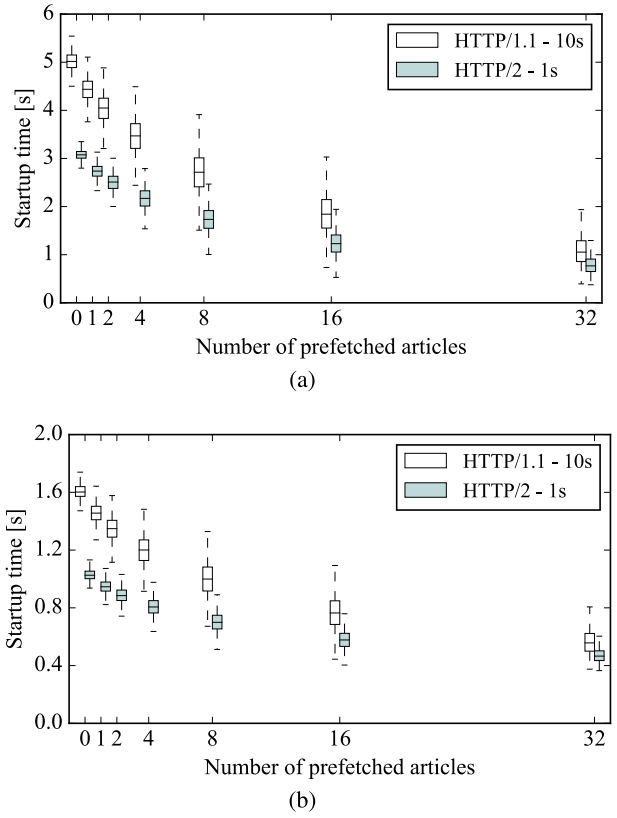

FIGURE 13. Startup time as a function of the number of prefetched articles, using a segment duration of 10 seconds over HTTP/1.1 or a segment duration of 1 second over HTTP/2. Boxplots include the average startup time for all 5,835 considered users. (a) 3G network. (b) 4G network.

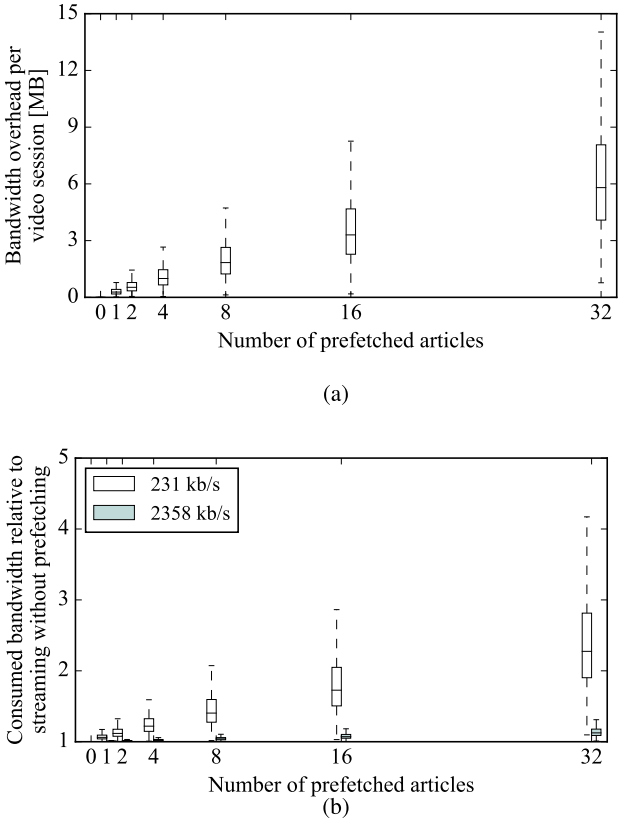

FIGURE 14. Bandwidth overhead as a function of the number of articles prefetched by the client. (a) Overhead per video session. (b) Estimation of the relative overhead.

We assume storage capacity is limited, so that previously stored content is removed when the list of articles is updated (e.g., when new articles have been published in the recency strategy, when a certain article generated more requests in the last hour in the popularity strategy, or when the user switches from one strategy to another, possibly resulting in a significantly different set of articles). For instance, the client might be able to store the top 8 video-based articles only, but not more.

The 5,835 users have sent a total of 1.8 million requests to video-based articles during the time of logging. Because of the time complexity, it is infeasible to replay all these requests for each evaluated parameter configuration. For this reason, the startup times for each of the videos in Figure 8 are used to represent the average startup time of each user. Of course, startup times are strongly dependent on the available bandwidth at the time of buffering, and can thus differ severely between sessions. Since the trace is started at a random point in time, however, the resulting values should be a close approximation of the expected startup time under realistic network conditions.

Figure 13 shows the average startup time as a function of the number of articles the client is able to prefetch. Naturally, the more articles the client can prefetch, the lower the average video startup time will be. As an example, the median startup time in a 3G network is reduced from 5.0 to 2.7 seconds (−45.9%) when the 8 most relevant videos for each user are prefetched with a segment duration of 10 seconds over HTTP/1.1. When this number is increased to 32, the median startup time can be reduced even further to 1.1 seconds (−78.0%). When a lower segment duration of 1 second is used and HTTP/2's server push is enabled, the startup time can be reduced from 3.1 to 1.7 (−43.5%) and 0.8 seconds (−74.1%) for 8 and 32 prefetched videos respectively. In a 4G scenario, the startup time for a segment duration of 10 seconds over HTTP/1.1 is reduced from 1.6 to 1.0 (−37.7%) and 0.8 seconds (−64.6%) for 8 and 32 prefetched videos respectively, and from 1.0 to 0.7 (−31.8%) and 0.5 seconds (−54.0%) for a segment duration of 1 second over HTTP/2.

Naturally, prefetching the content comes with a drawback: articles which are never requested, inevitably result in bandwidth overhead. Depending on the network carrier and the type of subscription, this can be detrimental to the user. In this context, Figure 14a shows the total overhead per video streaming session as a function of the number of videos the client is able to prefetch. When the 8 most relevant videos are prefetched, for instance, the median overhead amongst all users is 1.8 MB per video session, while the 90% percentile equals 3.6 MB. As a reference, downloading the

home page or a general single text-based article with JS, CSS and images included, requires 3.2 and 1.9 MB respectively. These numbers are in the same order of magnitude, and therefore, we conclude that a value of 8 is an ideal number of videos to prefetch by the client. Note that the required storage capacity is limited: the client only needs to be able to store the MPDs and initialization segments of eight videos, along with 8 $d_n = 8 \cdot 10$ seconds $= 80$ seconds of content at the lowest video quality.

Next to the absolute amount of wasted bandwidth, it would be interesting to determine the bandwidth overhead relative to the bandwidth consumed without prefetching. However, since the dataset does not include requests for single video segments, it is not possible to extract the exact time each user spends watching the content, nor what the total bandwidth consumption was at the time. Furthermore, the total bandwidth usage depends on all issued requests, including requests for the home page and text-based article requests. As a rough estimation, we can however assume that the user finishes each video, and that the user either streams the whole session at the average bitrate of the lowest quality (i.e., 231 kb/s), or at the average bandwidth in the provided 3G traces (i.e., 2358 kb/s). For each user, we keep track of the requested video, and determine its length based on the provided MPD files. Multiplying the total video length with the average bitrate results in the total bandwidth consumption for video content - discarding other resources such as JS, CSS and images. As illustrated in Figure 14(b), the relative overhead differs significantly for the two examples. When the 8 most relevant videos are prefetched, for instance, the median overhead is 40.4% in the former case, and 4.0% in the latter. In the end, the relative overhead strongly depends on the viewing behavior of the user, the videos considered and the type of network conditions during the video streaming session.

G. IMPACT ON BUFFER STARVATION

Reducing the number of bytes to transfer is an efficient way to reduce the video startup time. However, as mentioned previously in Section III-A, reducing the segment duration also results in lower buffer filling at the start of the video playout. When the available bandwidth is insufficient to provide the video at the lowest quality, or when the segment duration is changed too abruptly, this can result in buffer starvation and therefore, in playout freezes. To evaluate the impact of the proposed optimizations on buffer starvation, the 13,636 videos in the dataset with a minimum length of one minute, have been used to evaluate six different configurations. Similar as in the previous evaluations, each video is started at a random point in the provided throughput traces (but at the same time for all configurations) and the first minute of content is played out completely. Because we are interested in the impact of the configurations only, the content is played out at the lowest quality: this way, the applied rate adaptation heuristics in the DASH.js player do not come into play. Note that the stalling threshold in the player, defined as the amount of content (in seconds) which should be left in the buffer in order to continue playout, is set to 0 (i.e., play out as soon as content is available, as long as content is available).

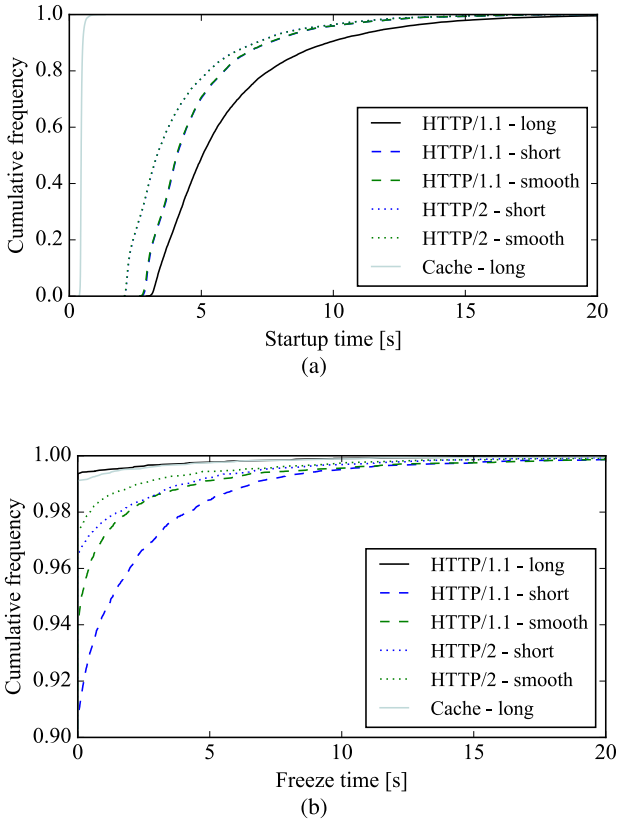

FIGURE 15. Cumulative distribution of the measured startup times and total freeze times for different configurations for the considered 13,636 videos. (a) Startup time. (b) Freeze time.

Figure 15 shows the cumulative distribution of the gains in measured startup time (top) and the total freeze time observed during playout of the first one minute of video (bottom), for six different configurations:

- HTTP/1.1 - long: a segment duration of 10 seconds;
- HTTP/1.1 - short: a segment duration of 1 second during the first 10 seconds, 10 seconds during the remainder of the stream;
- HTTP/1.1 - smooth: a segment duration smoothly changing from 1 to 2, 5 and eventually 10 seconds (a change occurs after each 10 seconds);
- HTTP/2 - short: idem as for HTTP/1.1, but now with HTTP/2 server push for shorter segments ($k = 10$);
- HTTP/2 - smooth: idem as for HTTP/1.1, but now with HTTP/2 server push for shorter segments ($k = 10, 5, 2$ respectively);
- Cache - HTTP/1.1: the first 10 seconds are retrieved from local storage, the remainder of the session from the server.

Figure 15(a) shows the results which were previously obtained: the startup time can be reduced significantly when the segment duration is lowered, when HTTP/2 server push

is applied and when caching is used. Note that the ''short'' and ''smooth'' approaches result in the same startup time, since both require the same resources to start video playout (including a single one-second video segment).

Figure 15(b) shows the total freeze time observed during playout of the first minute of content. When the default configuration is used, only 0.6% of video streaming sessions suffers from playout freezes. When switching from the cache to the local server, a new TCP connection has to be started, which implies that the available bandwidth cannot immediately be used yet. For this reason, it can take a while before the second segment arrives, in cases of low bandwidth sometimes resulting in a playout freeze. When a segment duration of 1 second is used for the first 10 seconds of content, results show that the client is indeed more prone to buffer starvation: when using the ''short'' segment scheme, 9.5% of video streaming sessions results in playout freezes. Adopting the ''smooth'' scheme, this number is reduced to 6.1%. Applying HTTP/2 server push on top of that, a further reduction to 2.7% is achieved. It is worth noting that most rebuffering events do not last longer than 500 ms, which is significantly smaller than the gains in terms of startup time (a median reduction of 2.1 seconds, with outliers higher than 15 seconds). Higher values occur only in cases where the available throughput is significantly lower than the bitrate for the lowest quality level, in which case no approach can achieve a desirable result: one can argue that in the given use case, it can be better to start the stream under 10 seconds while temporarily suffering from rebuffering events, than starting the stream in 10 seconds or more and risking abandonment by the user.

H. SUMMARY

In the evaluations above, we showed that the proposed optimizations can result in significantly shorter video startup times, which is beneficial when browsing news content in a video web portal. In summary, the main reductions are achieved by:

1) Using a shorter video segment duration: when the available bandwidth is limited, this can result in reductions in the order of 10 to 25%. This optimization is straightforward to implement, as it requires minor adaptations to the MPD and limited additional storage at server-side;
2) Using HTTP/2 on the application layer, making use of HTTP/2 server push: even when the network delay is limited to 120 ms, this results in an additional reduction in the order of 15 to 25%. At server-side, this optimization requires a separate request handler to push required resources. Since most browsers nowadays have full support for HTTP/2, no changes are required at client-side;
3) Using server-side user profiling and client-side storage to prefetch (the first part of) possibly relevant videos: storing the 8 most relevant video articles, for instance, additional reductions in the order of 45% can be achieved. This however comes at the cost of increased bandwidth usage by the client. This optimization requires server-side logging and analytics, content prefetching and client-side storage, making it less straightforward to deploy.

All aforementioned optimizations are complementary: one can, for instance, consider a scenario where a segment duration of 1 second is used, HTTP/2 server push is enabled and up to 8 articles are prefetched on a per-user bases. Comparing results with the reference scenario for deredactie.be, i.e., a segment duration of 10 seconds over HTTP/1.1 without prefetching, the median startup time can be reduced from 5.0 to 1.7 seconds (−66.0%) in a 3G network scenario, and from 1.6 to 0.7 seconds (−56.3%) in a 4G network scenario.

V. CONCLUSIONS

In this work, a novel framework for low-latency delivery of news-related video content is presented. Its main components include server-side encoding, HTTP/2's server push, user profiling and client-side storage for proactive content delivery. Through a relevant use case of a major Belgian news provider, we showed that each of the proposed optimizations can significantly reduce the median startup time of video streaming sessions, by 10 to 25% using a shorter video segment duration, by 15 to 25% using HTTP/2 server push and by 45% when the first 10 seconds of the 8 most relevant videos are stored at client-side. Combining these optimizations, the median startup time can be reduced by more than 50% in both 3G and 4G mobile networks. These reductions allow the news provider to improve the user's Quality of Experience, encouraging low-latency user interaction with the provided video content. In future work, we will characterize the performance of the considered optimizations in other use case scenarios, such as 360° video delivery for virtual reality.

REFERENCES

[1] S. Egger, T. Hoßfeld, R. Schatz, and M. Fiedler, ''Waiting times in quality of experience for web based services,'' in *Proc. Int. Workshop Qual. Multimedia Exper.*, 2012, pp. 86–96.
[2] 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 Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 469–492, 1st Quart., 2015.
[3] R. Mok, E. Chan, and R. Chang, ''Measuring the quality of experience of HTTP video streaming,'' in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, May 2011, pp. 485–492.
[4] T. Stockhammer, ''Dynamic adaptive streaming over HTTP standards and design principles,'' in *Proc. ACM Conf. Multimedia Syst.*, 2011, pp. 133–144.
[5] T. Hoßfeld, S. Egger, R. Schatz, M. Fiedler, K. Masuch, and C. Lorentzen, ''Initial delay vs. interruptions: Between the devil and the deep blue sea,'' in *Proc. Int. Workshop Qual. Multimedia Exper.*, 2012, pp. 1–6.
[6] L. Chen, Y. Zhou, and D. M. Chiu, ''Video browsing—A study of user behavior in online VoD services,'' in *Proc. Int. Conf. Comput. Commun. Netw.*, 2013, pp. 1–7.
[7] J. van der Hooft, C. De Boom, S. Petrangeli, T. Wauters, and F. De Turck, ''An HTTP/2 push-based framework for low-latency adaptive streaming through user profiling,'' in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp.*, Apr. 2018, pp. 1–5.

[8] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," in *Proc. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.
[9] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 197–210.
[10] Y. S. de la Fuente *et al.*, "iDASH: Improved dynamic adaptive streaming over HTTP using scalable video coding," in *Proc. ACM Conf. Multimedia Syst.*, 2011, pp. 257–264
[11] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 9, pp. 1103–1120, Sep. 2007.
[12] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.
[13] A. Deveria. (2018). *Can I Use HEVC?*. [Online]. Available: https://caniuse.com/#search=HEVC
[14] Y. Shuai and T. Herfet, "On stabilizing buffer dynamics for adaptive video streaming with a small buffering delay," in *Proc. IEEE Consum. Commun. Netw. Conf.*, Jan. 2017, pp. 435–440.
[15] K. Miller, A. Al-Tamimi, and A. Wolisz, "QoE-based low-delay live streaming using throughput predictions," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 13, no. 1, pp. 4:1–4:24, 2016.
[16] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive," *IEEE/ACM Trans. Netw.*, vol. 22, no. 1, pp. 326–340, Feb. 2014.
[17] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan, "Towards a SPDY'Ier mobile Web?" in *Proc. ACM Conf. Emerg. Netw. Exp. Technol.*, 2013, pp. 303–314.
[18] Y. Elkhatib, G. Tyson, and M. Welzl, "Can SPDY really make the Web Faster?" in *Proc. IFIP Netw. Conf.*, 2014, pp. 1–9.
[19] X. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How Speedy is SPDY?" in *Proc. USENIX Conf. Netw. Syst. Design Implement.*, 2014, pp. 387–399.
[20] S. Wei and V. Swaminathan, "Low latency live video streaming over HTTP 2.0," in *Proc. Netw. Oper. Syst. Support Digit. Audio Video Workshop*, 2014, pp. 37:37–37:42.
[21] W. Cherif, Y. Fablet, E. Nassor, J. Taquet, and Y. Fujimori, "DASH fast start using HTTP/2," in *Proc. ACM Workshop Netw. Oper. Syst. Support Digital Audio Video*, 2015, pp. 25–30
[22] K. Gatimu, A. Dhamodaran, T. Johnson, and B. Lee, "Experimental study of low-latency HD VoD streaming using flexible dual TCP-UDP streaming protocol," in *Proc. Consum. Commun. Netw. Conf.*, 2018, pp. 1–6.
[23] W3C/IETF. (2018). *Web Real-Time Communication (WebRTC)*. [Online]. Available: https://www.webrtc.org
[24] *Dynamic Adaptive Streaming Over HTTP (DASH)—Part 5: Server and Network Assisted DASH (SAND)*, document ISO/ICE 23009-5:2017, 2017. [Online]. Available: https://www.iso.org/obp/ui#iso:std:iso-iec:23009:-5:ed-1:v1:en
[25] A. Bentaleb, A. C. Begen, and R. Zimmermann, "SDNDASH: Improving QoE of HTTP adaptive streaming using software defined networking," in *Proc. ACM Multimedia Conf.*, 2016, pp. 1296–1305.
[26] S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck, "Software-defined network-based prioritization to avoid video freezes in HTTP adaptive streaming," *Int. J. Netw. Manage.*, vol. 26, no. 4, pp. 248–268, 2016.
[27] D. K. Krishnappa, M. Zink, C. Griwodz, and P. Halvorsen, "Cache-centric video recommendation: An approach to improve the efficiency of YouTube caches," *ACM Trans. Multimedia Comput., Commun. Appl.*, vol. 11, no. 4, pp. 48:1–48:20, 2015.
[28] M. Claeys, N. Bouten, D. De Vleeschauwer, W. Van Leekwijck, S. Latré, and F. De Turck, "Cooperative announcement-based caching for video-on-demand streaming," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 2, pp. 308–321, Jun. 2016.
[29] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri, "Bandwidth-aware prefetching for proactive multi-video preloading and improved HAS performance," in *Proc. ACM Multimedia Conf.*, 2015, pp. 551–560.
[30] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Comput.*, vol. 42, no. 8, pp. 30–37, Aug. 2009.
[31] C. De Boom *et al.*, "Large-scale user modeling with recurrent neural networks for music discovery on multiple time scales," *Multimedia Tools Appl.*, vol. 77, no. 12, pp. 15385–15407, Jun. 2018.
[32] J. Basilico and Y. Raimond, "Déjà Vu: The importance of time and causality in recommender systems," in *Proc. Conf. Recommender Syst.*, 2017, p. 342.
[33] T. Donkers, B. Loepp, and J. Ziegler, "Sequential user-based recurrent neural network recommendations," in *Proc. Conf. Recommender Syst.*, 2017, pp. 152–160.
[34] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *CoRR*, vol. abs/1511.06939, Mar. 2015. [Online]. Available: https://arxiv.org/abs/1511.06939
[35] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck, "An HTTP/2 push-based approach for low-latency live streaming with super-short segments," *J. Netw. Syst. Manage.*, vol. 26, no. 1, pp. 51–78, 2018.
[36] I. Sodagar, "The MPEG-DASH standard for multimedia streaming over the Internet," *IEEE Multimedia*, vol. 18, no. 4, pp. 62–67, Nov. 2011.
[37] S. Van Canneyt, B. Dhoedt, S. Schockaert, and T. Demeester, "Knowledge extraction and popularity modeling using social media," Ghent Univ.-IMEC, Ghent, Belgium, Tech. Rep., 2016. [Online]. Available: https://biblio.ugent.be/publication/8133560
[38] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, Jan. 2013. [Online]. Available: https://arxiv.org/abs/1301.3781
[39] nPerf. (2018). *Cellular Data Networks in Belgium*. [Online]. Available: https://www.nperf.com/en/map/BE/-/-/signal/
[40] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Video streaming using a location-based bandwidth-lookup service for bitrate planning," *ACM Trans. Multimedia Comput., Commun. Appl.*, vol. 8, no. 3, pp. 24:1–24:19, 2012.
[41] J. van der Hooft *et al.*, "HTTP/2-based adaptive streaming of HEVC video over 4G/LTE networks," *IEEE Commun. Lett.*, vol. 20, no. 11, pp. 2177–2180, Nov. 2016.

JEROEN VAN DER HOOFT received the M.Sc. degree in computer science engineering from Ghent University, Belgium, in 2014. In 2014, he joined the Department of Information Technology, Ghent University-imec, where he is currently pursuing the Ph.D. degree. His main research interest is the end-to-end quality of experience optimization in adaptive video streaming.

CEDRIC DE BOOM received the M.Sc. and Ph.D. degrees in computer science engineering from Ghent University-imec, Belgium, in 2014 and 2018, respectively. In 2016, he has been involved in the topic of music discovery models at Spotify Inc. His main research interests lie in information retrieval and machine learning applications, more specifically in user modeling and text processing. In 2015, he started focusing on deep learning applications in recommender systems and sequence modeling.

STEFANO PETRANGELI received the M.Sc. degree in systems engineering from the Sapienza University of Rome in 2011, the Second Level master's degree from Telecom Italia Laboratories and the Polytechnic of Turin in 2013, and the Ph.D. degree in computer science engineering from Ghent University, Belgium, in 2018. His main research interest is the end-to-end quality of experience optimization of internet video streaming delivery, with a particular focus on immersive media.

TIM WAUTERS received the M.Sc. and Ph.D. degrees in electrotechnical engineering from Ghent University, Belgium, in 2001 and 2007, respectively. He is currently a Post-Doctoral Fellow with the Department of Information Technology, Ghent University-imec. His work has been published in over 100 scientific publications in international journals and in the proceedings of international conferences. His research interests include network and service architectures, and management solutions for multimedia delivery services.

FILIP DE TURCK received the M.Sc. and Ph.D. degrees in electronic engineering from Ghent University, Belgium, in 1997 and 2002, respectively. He is currently a Full Professor with Ghent University-imec, where he leads the Network and Service Management Research Group at the Department of Information Technology. His research interests include telecommunication network and service management, and design of efficient virtualized network systems. He serves as the Chair for the IEEE Technical Committee on Network Operations and Management. He is the Editor-in-Chief of the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT.