Bandwidth-aware Prefetching for Proactive Multi-video Preloading and Improved HAS Performance

Vengatanathan Krishnamoorthi[†] Niklas Carlsson[†] Derek Eager[‡]
Anirban Mahanti[§] Nahid Shahmehri[†]
[†] Linköping University, Sweden, firstname.lastname@liu.se
[‡] University of Saskatchewan, Canada, eager@cs.usask.ca
[§] NICTA, Australia, anirban.mahanti@nicta.com.au

ABSTRACT

This paper considers the problem of providing users playing one streaming video the option of instantaneous and seamless playback of alternative videos. Recommendation systems can easily provide a list of alternative videos, but there is little research on how to best eliminate the startup time for these alternative videos. The problem is motivated by services that want to retain increasingly impatient users, who frequently watch the beginning of multiple videos, before viewing a video to the end. We present the design, implementation, and evaluation of an HTTP-based Adaptive Streaming (HAS) solution that provides careful prefetching and buffer management. We also present the design and evaluation of three fundamental policy classes that provide different tradeoffs between how aggressively new alternative videos are prefetched versus the importance of ensuring high playback quality. We show that our solution allows us to reduce the startup times of alternative videos by an order of magnitude and effectively adapt the quality such as to ensure the highest possible playback quality of the video being viewed. By improving the channel utilization we also address the discrimination problem that HAS clients often suffer from, allowing us to in some cases simultaneously improve the playback quality of the video being viewed and provide the value-added service of allowing instantaneous playback of the prefetched alternative videos.

Categories and Subject Descriptors

C.4 [**Information Systems Organization**]: Performance of Systems; C.2.2 [**Network Protocols**]: Applications; H.5.1 [**Multimedia Information Systems**]: Video

Keywords

HTTP-based adaptive streaming (HAS); Bandwidth-aware prefetching; Multi-video preloading; Seamless playback

MM'15, October 26–30, 2015, Brisbane, Australia.

DOI: http://dx.doi.org/10.1145/2733373.2806270.

1. INTRODUCTION

Today's users are highly impatient and frequently switch movies or channels within a few minutes of viewing. For example, analysis of 540 million video-on-demand sessions has shown that users often view the beginning of a few videos (5 on average) before finally viewing one video to the end [6]. To retain users and help minimize the chance that users decide to use competing providers, it is therefore important to quickly provide these users with attractive alternative video choices that they can view next. For example, a selection of alternative videos could be presented to the user as thumbnails, or the user could be presented previews or ads whenever pausing playback. We argue that to maximize retention rates, these alternative videos ideally should be proactively preloaded or prefetched to the client in a manner that allows the videos to be available for instantaneous viewing at the time when the client decides to terminate playback of the originally viewed video or to switch video to view.

While there is much work on recommendation algorithms for determining the best set of alternative videos that the user is likely to want to view next [7, 8, 28], there is a lack of research on the best ways to prefetch these alternative videos. In order to maximize the user perceived Quality of Experience (QoE), such techniques must balance the importance of high-quality uninterrupted playback against the desire to allow instantaneous playback of the alternative videos. This problem is both important and challenging, as users often are highly heterogeneous with time-varying bandwidth conditions and the alternative videos must be carefully prefetched in parallel with the video being streamed such as to ensure timely preloading of these videos, while at the same time minimizing the impact on the streamed video.

In this paper we present the design, implementation, and evaluation of an HTTP-based Adaptive Streaming (HAS) solution that provides prefetching and buffer management such as to ensure instantaneous playback of alternative videos whenever a user selects one of the alternative videos or interrupts playback. The use of HAS allows the quality of both the streaming video (currently being viewed) and the prefetched alternative videos (that are cached for potential future viewing) to be adapted such as to make the best use of the available bandwidth. Within our framework, carefully designed prefetching policies are used to ensure interruption free playback, while allowing the player different (policy dependent) means to control when each alternative video is expected to be ready for instantaneous playback.

Typical HAS players are designed to interrupt download when the buffer is sufficiently full, as measured by an upper

buffer threshold T_{max}, for example, and not resume download again until the buffer has drained to some lower buffer threshold T_{min}. While the use of an upper threshold T_{max} helps limit the wasted resources associated with users not watching the full video, it has been shown that the resulting *on-off behavior* [5, 21, 24] can lead to poor performance under conditions with competing traffic [3, 13]. Our solution carefully schedules prefetching during off-periods, obtaining improved download rates, while also ensuring that the prefetching does not negatively impact the user playback experience. In fact, we show that under some circumstances, prefetching of alternative videos can even improve the playback quality of the streaming video itself.

It is likely that the criticality of preloading alternative videos will differ both across services and between users within a service. For example, for some users, timely preloading of alternative videos may be very important such as to allow seamless "video browsing", whereas for other users that are less likely to change videos, the preloading is simply a value added service used occasionally.

Motivated by a diverse set of use cases, and user behaviors, within our framework, we present the design and evaluation of three fundamental policy classes used for multi-file prefetching. The first two policy classes use opportunistic prefetching only during off periods, but differ in how aggressively they prefetch new alternative videos versus ensuring that the prefetched videos are relatively evenly paced. The third class uses strict prefetch deadline targets for when the beginning of the different alternative videos should have been prefetched. In bandwidth restricted scenarios, this policy class must therefore sometimes trade a lower quality level of the streaming video for more reliable delivery of the prefetched alternative videos. We use an optimization framework to balance the importance of the different deadlines associated with this policy. Within each policy class, we consider both non-adaptive and adaptive quality selection versions.

We implement a proof-of-concept solution on top of the Open Source Media Framework (OSMF), within which we combine the OSMF Ad insertion plugin (to allow playback of multiple videos in the same video container) and a download-manager extension based on Krishnamoorthi et al.'s [19] branched video framework (that supports prefetching and caching). Our implementation allows the user to seamlessly switch to viewing an alternative video with negligible startup delay, as soon as the player has prefetched the beginning of the alternative video(s). Moreover, the use of the ad plugin to stitch together multiple videos within a single container and cache management allow the currently played video to be quickly replaced by an already prefetched video. In experiments assuming a 4 Mb/s link, for example, the startup time of an alternative video (at lowest playback quality) is reduced from about 6 seconds (without prefetching but using the ad plugin for faster playout) to 0.6 seconds (needed to switch and load containers).

In summary, the primary contributions of the paper are (i) a prefetching framework that simultaneously allows adaptive prefetching and playback of alternative videos, while addressing the on-off problem typically associated with HAS, (ii) a novel proof-of-concept implementation that we show implements effective prefetching policies and provides close to instantaneous playback of alternative videos, and (iii) the design and evaluation of three candidate policy classes, including both non-adaptive and quality adaptive versions. Our experimental results highlight and validate the basic properties associated with the different policy classes and characterize their performance tradeoffs.

The remainder of the paper is organized as follows. Sections 2 and 3 set the context and describe our system framework, respectively. We then define the policy classes and their variations (Section 4), describe our system implementation (Section 5), and present our validation and performance results (Section 6). Finally, related work (Section 7) and conclusions (Section 8) are presented.

2. BACKGROUND

HTTP-based Adaptive Streaming (HAS) is the most popular means of providing streaming services today [2]. HAS is client driven and allows the use of stateless Web servers and caches. Typically, the players either use byte-range requests or request each individual file "chunk" with a unique URL [5]. The use of multiple files with different video qualities, with aligned chunks, allows the player to adapt the playback media streaming rate based on current buffer and network conditions.

Commonly, the client-side scheduling of chunk requests by HAS players results in on-off periods in the bandwidth usage [24]. To understand why, note that players must balance the need to buffer sufficiently many chunks such as to account for jitter and bandwidth variations against the desire to avoid wasting unnecessary resources on clients terminating playback early. For these reasons, HAS players typically try to maintain some minimum desired buffer occupancy T_{min} (measured in second of playback) without exceeding some maximum desired buffer occupancy T_{max}. For example, with the simplest policies, as used by the OSMF player, for example, playback begins whenever T_{min} data is buffered and the player continues to make requests until T_{max} is reached. Then, the player does not make any new requests until the buffer occupancy falls below T_{min} again.

Similar on-off patterns have been observed with other players [5, 21, 24]. One noticeable difference between players is the size of the buffer thresholds. For example, among general video streaming frameworks, OSMF uses much smaller default values ($T_{min}/T_{max} = 4/6$) compared to Microsoft Smooth Streaming ($T_{min}/T_{max} = 12/30$) [5]. Even more extreme is the $T_{max} = 240$ second buffer that Netflix tries to maintain in steady state [14]. A larger buffer provides additional protection against errors in rate estimation, and can in some cases allow clients to recover from temporary network outages. Note that OSMF may be designed for short duration videos played on mobile devices, whereas Netflix focuses on delivering full-length movies and TV shows. Yet, despite the large differences in the desired buffer size, the on-off behavior is common across the players.

Coupled with TCP congestion control mechanisms, the on-off periods can result in clients not obtaining their fair bandwidth share [13, 15]. To understand why, note that the overall throughput of HAS depends on the underlying end-to-end congestion mechanism of TCP. For example, consider the case in which the congestion window timer times out during an off period. In this case, the sender will reset its congestion window to a predefined minimum and begin slow-start at the start of the on-period. More generally, the use of on-off periods can cause a user to pick a lower quality than it otherwise would, resulting in increasingly bigger

off-periods, thus allowing other users to grab a bigger bandwidth share. This behavior is self-feeding and HAS users can get increasingly smaller bandwidth share [13].[1]

In this paper, we present a novel system design that utilizes the otherwise unused bandwidth during off periods to prefetch the initial chunks of alternative videos. This allows the player to simultaneously preload alternative videos and maintain its fair bandwidth share.

3. SYSTEM OVERVIEW

3.1 Basic Approach

We assume that each user has been assigned a set of alternative videos that the content provider would like to preload at the client. This list could include, for example, videos that the user is likely to view next or ads that the provider may want to show when the user pauses playback. We will refer to the video being streamed as the *streaming video*, and the videos to be preloaded as *alternative videos*. To allow instantaneous playback of an alternative video, its initial chunks must be prefetched and cached locally on the client.

Our solution leverages HAS chunking and adaptivity. To allow instantaneous playback of alternative videos, the initial chunks of these videos are downloaded in parallel with the chunks of the currently streamed video. By prefetching the initial chunks of the alternative videos, the player can allow the user to seamlessly switch between multiple videos without enduring noticeable playback interruption. To ensure interruption free streaming of the currently viewed video, the video qualities of both the streamed video and of the prefetched alternative videos are adapted based on current conditions.

Our solution utilizes the off periods associated with typical HAS players to perform prefetching. Using opportunistic and adaptive prefetching of the alternative videos during time periods when connections otherwise would be idle allows us to make use of the otherwise wasted bandwidth. By eliminating off periods, the player also avoids TCP congestion window timeouts and other throughput problems associated with the on-off pattern of typical HAS systems, allowing the client to maintain its fair bandwidth share and improve download speeds.

In Figure 1(a) we use example traces to illustrate how prefetching of alternative videos can help improve the client's overall download throughput during on periods, and hence also the playback quality of the streaming video itself. Note that after the off period (during which the regular HAS player would not be downloading chunks), there is a non-negligible ramp-up period required by TCP before the regular HAS player reaches the original download rate, resulting in a slower average download rate during the following on period. This is clearly illustrated by the difference in the slope of the cumulative download curves (Figure 1(b)) between 28 and 37 seconds. The higher download rate also results in T_{max} being reached sooner, illustrated by the streaming video curve of our player plateauing three seconds before the native HAS player without prefetching.

[1]While the impact of on-off periods differs between TCP implementations, typically TCP enters slow start when the idle period exceeds the RTO threshold. Section 6.7 takes a closer look at the impact of TCP version and slow start.

3.2 Simple Prefetch Example

Figure 2 illustrates the operation of a basic "best effort" prefetching policy that prefetches the first two chunks of each of the two alternative videos. In this example there are only two quality encodings, and the player downloads the initial chunks of the alternative videos, at the lower quality, whenever there would otherwise be an off period.

As with regular HAS, the client starts downloading the first few chunks of the streaming video at a low quality and then increases quality as network conditions permit. The player continues to download chunks back-to-back until the buffer occupancy reaches or exceeds T_{max}. At this time, rather than entering an off period, our modified player now downloads the chunks of the alternative video, starting with the first chunk of the first alternative video.

Chunks of the alternative videos are downloaded into a separate cache and are not loaded directly into the player. Therefore, the playback buffer is drained at exactly the same rate as for a regular HAS player. The modified player continues downloading alternative chunks until the next expected completion time would be after the playback buffer reaches T_{min}. For example, in the case that the current download rate is r and the current playback buffer is T, the player should no longer initiate new downloads of alternative chunks whenever $r(T - T_{min})$ is smaller than the size S of the next alternative chunk. Instead, the player switches back to downloading chunks from the streaming video, and the download pattern is repeated.

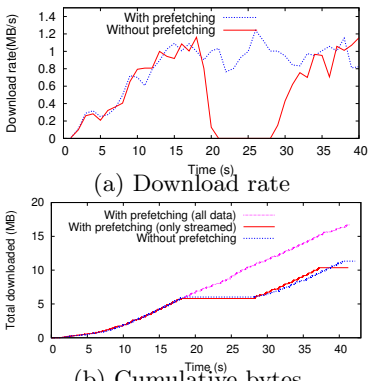
4. PREFETCH POLICIES

All considered prefetch policy classes (Section 4.1) assume an ordered list of alternative videos, but differ with respect to how many chunks are downloaded of each alternative video, the timing of when alternative video chunks are downloaded, and how strictly potential chunk deadlines are applied. Variants of the policy classes (Section 4.3) are defined based on their quality adaptation.

4.1 Policy Classes

Best-effort: This is the simplest policy class. When the buffer occupancy reaches T_{max}, the player starts downloading chunks of the alternative videos, and continues doing so until it estimates the download rate to be insufficient to allow another such chunk download before the buffer falls below T_{min}. At this time, the player goes back to downloading chunks of the streaming video. When given a prefetch opportunity, this policy always prefetches the first n chunks of the alternative video next on the list of alternative videos, one after the other. Here, n is a policy parameter controlling how much of each video should be prefetched before being presented as a preloaded alternative.

This policy class is well suited for network conditions where the buffer size periodically reaches T_{max}. Naturally, T_{max} is reached most frequently under high-bandwidth scenarios. However, interestingly, due to the quality adaptation of typical HAS players, alternative chunks will also be downloaded relatively frequently also under low bandwidths scenarios, as long as the download rate exceeds the minimum encoding rate. Perhaps the biggest downside with this policy class is that it does not provide the content provider with any control of how quickly new alternative videos are prefetched. The following two policy classes address this shortcoming.



(a) Download rate

(b) Cumulative bytes

Figure 1: Transfer during example experiment.

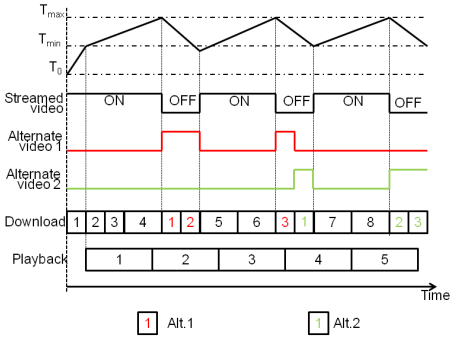


Figure 2: Prefetch framework overview.

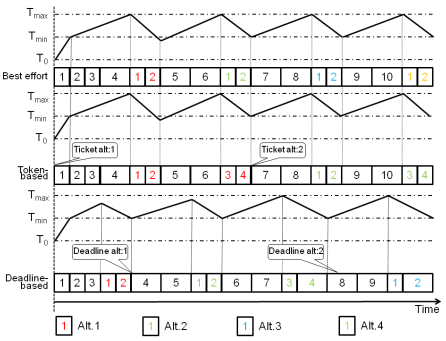


Figure 3: High-level comparison of prefetching classes.

Token-based: This policy class provides greater control of the earliest time new alternative videos should be prefetched, allowing the client to pace the rate that such videos are prefetched. These policies download chunks from the alternative videos at exactly the same instances as the *best-effort* policies, but differ in how they decide from which video to download next. To achieve an even pacing without impairing the playback quality of the streaming video, these policies implement a token-bucket approach in which new video tokens are generated and placed in a queue every Δ seconds. Each token is associated with a new alternative video, and the player always downloads the first n chunks of the video next in the token-bucket queue, unless the queue goes empty, in which case the player continues to select chunks of the video associated with the most recent token. The policy parameter Δ determines the pacing, and n the minimum amount of prefetching of each video. With $\Delta = 0$, the token-based policy is equivalent to best-effort.

Except for a much more even pacing of alternative videos, the performance of the *token-based* policies is the same as for the corresponding *best-effort* policies. Both these policy classes only download alternative video chunks when given an opportunity to do so without adapting the downloads of the streaming video, ensuring that the playback quality of the streaming video is not negatively impacted.

Deadline-based: For some video browsing users [6], stall-free switching between videos may be more important than achieving the highest possible streaming rate of the currently streamed video. To further prioritize the download of alternative videos and enforce an even pacing of the alternative videos, the third policy class associates deadlines with each alternative video and applies an optimization framework to find the best tradeoff between video quality and ensuring that the alternative videos are prefetched in time of their respective deadlines.

For simplicity, we define relative playback deadlines, such that the deadline of the next alternative video a always occurs when m_a chunks of the streaming video have been played either since the deadline for the previous alternative video or since playback began (in the case there are no prior deadlines). At each such deadline the player is expected to have downloaded the first n chunks of the alternative video. Assuming that the m_a chunks of the streaming video are downloaded ahead of the n chunks of the alternative video, we can now determine an optimized download schedule by solving the following optimization problem:

$$\text{maximize} \sum_{i=1}^{m_a+1} q_i^s l_i^s + \sum_{j=1}^{n} q_j^a l_j^a, \tag{1}$$

subject to

$$t_i^s \le \tau + \sum_{j=1}^{i-1} l_j^s, \quad 1 \le i \le m_a + 1 \tag{2}$$

$$t_j^a \le \tau + \sum_{i=1}^{m_a} l_j^s, \qquad 1 \le j \le n \tag{3}$$

$$r_i^s (t_i^s - t_{i-1}^s) = q_i^s l_i^s, \quad 2 \le i \le m_a + 1 \tag{4}$$

$$r_1^a (t_1^a - t_{m+1}^s) = q_1^a l_1^a, \tag{5}$$

$$r_j^a (t_j^a - t_{j-1}^a) = q_j^a l_j^a, \qquad 2 \le j \le n \tag{6}$$

The structure of this optimization model is similar to that used by Krishnamoorthi et al. [19] to address a different prefetching problem, concerning "branched video". Here, q_i^s, l_i^s, and t_i^s are the quality, chunk size, and download completion time of the i^{th} chunk of the streaming video; q_j^a, l_j^a and t_j^a are the corresponding values for the next alternative video. Furthermore, r_i^s and r_j^a are the estimated download rates for chunks i and j of the two videos, respectively, and the startup delay τ is equal to the most recent deadline, or the playback start of the first downloaded chunk.

The objective function (1) tries to maximize the playback quality of the streaming video, conditioned on the individual playback deadlines of the streaming video (2), the n alternative chunks being downloaded before their deadline (3), and bandwidth flow conservation assuming that the n alternative chunks (5 and 6) are downloaded after the m_a chunks of the streaming video (4). This prioritization ensures that the client does not experience unnecessary stall times for the streaming video. However, as chunks from alternative videos may be prefetched even when the player is not in an off-period, it may potentially impair the playback quality of the streaming video.

Whenever the deadline of an alternative video is passed, we simply move on to the next deadline. Furthermore, in the case that n chunks of the alternative video have been downloaded but the deadline of the alternative video is not passed, chunks from the streaming and alternative videos are downloaded in round-robin order. This approach therefore paces the downloads of alternative videos and naturally limits the workahead of the streaming video.

4.2 High-level Comparison

Figure 3 presents a high-level comparison between the three policy classes. In this example, we have assumed $n = 2$ and that the prefetched chunks are of the lower of two qualities. We note that the best-effort policy consistently downloads the first two chunks of each alternative video, whereas the token-based policy uses exactly the same opportunities to build up a larger buffer (workahead) for the video associated with the tokens generated thus far. This illustrates the differing policy tradeoffs between prioritizing pacing the prefetching of new alternative videos (potentially building up additional workahead for the current video that the client may be more likely to start viewing) and prefetching the largest possible set of alternative videos.

The biggest differences are observed for the deadline-based policy class. In contrast to the first two policy classes, this policy class uses deadlines to enforce downloads of alternative videos. Of course, this can come at the expense of the streaming video itself. For example, in Figure 3 the more aggressive prefetching required to meet the download deadlines of the alternative video chunks results in earlier downloads of these chunks and the playback buffer of the streaming video does not reach T_{max} until after the first such deadline. Note also that the more aggressive use of prefetching results in tighter buffer conditions. As seen in our experimental results (Section 6), this often results in the streaming video being streamed at a lower overall average quality compared to with the other two policies.

4.3 Fixed vs. Adaptive Prefetching

Thus far, for each policy class, we have described a simple policy variant that always prefetches chunks of the alternative videos at the lowest quality. We call these policy variants *lowest quality*. We also consider variants that adapt the quality of the alternative chunks based on the current buffer conditions and download rates. With our *adaptive quality* variants the quality of each prefetched chunk is selected at the time of the download request. At this time, the client greedily maximizes the quality of the next prefetched chunk, based on the estimated bandwidth and conditioned on the download being completed before the buffer occupancy falls below T_{min} and any other download deadline constraints associated with the *deadline-based* policy, for example.

5. SYSTEM IMPLEMENTATION

We have implemented a prototype of our system on top of the OSMF framework. Although the general policies described here are applicable to any HAS system, one advantage of the OSMF framework is that it is open source and we can share our implementation with the research community.[2] To simplify the head-to-head comparison of the prefetching policy classes, and to help ensure that our results are not determined by some quirk of OSMF, we use a somewhat simplified rate switching algorithm which does not leverage OSMF specific rules such as `DroppedFPSRule` or `EmptyBufferRule`. This algorithm consistently determines the quality of the next chunk as the highest chunk encoding rate that is lower than 0.8 times the current estimated bandwidth. An EWMA on the chunk download rates with α=0.4 is used to estimate the current download rate. The specific rate switching algorithm used was found to not significantly impact the relative policy comparisons.

The prefetching policies are implemented in a separate `HTTPDownloadManager` class that we incorporate in the OSMF framework. To facilitate seamless playback of multiple separate videos within the same container (e.g., to show snippets) we leverage the OSMF Ad insertion plugin.

Multi-file prefetching: Our `HTTPDownloadManager` class is responsible for (i) estimating download rates, (ii) updating buffer occupancies, (iii) regulating on-off periods, and (iv) managing the download schedule of chunks from the streaming video and alternative videos. This class integrates with the original OSMF libraries, thereby facilitating transparent operations between the original and custom components of the player. Chunks which are downloaded by this class are placed in the browser cache. When initiating playback of an alternative video these chunks are then requested by the player, resulting in a browser cache hit, from which the chunks can be quickly retrieved and subsequently played.

Browser cache: In our experiments, we make use of the browser cache as a temporary store for prefetched data rather than holding this data internally within the player. This circumvents limitations within the OSMF platform. For our experiments we place the browser cache on the RAM as it provides the fastest retrieval times. In the case a larger cache is needed, the cache may also be placed on the hard disk. The difference between these two configurations was verified to be minimal in our system.

Multi-file playback: The OSMF Ad insertion plugin was originally designed to play advertisements without having to load a new player/webpage each time, and allows multiple separate videos to be played within the same video container. We modified this plugin to allow different videos to be seamlessly stitched together at the time instance when the user selects to view the alternative video.

User interaction: There are multiple ways for the alternative videos to be displayed to the user. First, the user can be presented with a list of alternative videos that currently have been prefetched and are available for instantaneous playback. As new alternative videos are prefetched, they are added to the list. Second, when the streaming video is paused, the container can be replaced with a different instance of the video player, which plays one or more alternative videos. In this case, upon completion, playback of the original streaming video is automatically resumed.

The plugin also supports overlaying videos to be displayed. In this case, both the streaming video currently being viewed by the user, and one or more alternative videos can be displayed simultaneously in the same container. Other content providers may want to just display a thumbnail image or example frame of the alternative video. In this paper, we focus on the performance aspects of our framework, and leave user interface design issues for future work.

6. VALIDATION AND PERFORMANCE

6.1 Experimental Setup

Our experiments have been performed using a high-speed LAN. Dummynet [26] is used to control available bandwidths and round-trip times (RTTs). Our videos are hosted on a Windows 7 machine running Flash Media Server (FMS) 4.5 on top of TCP Reno. We also cross validate our results with FMS running on a Linux machine with TCP CUBIC

[2]Our data, source code, and system framework are available at `http://www.ida.liu.se/~nikca89/papers/mm15.html`.

and TCP Reno as the congestion control algorithm. The client machine runs Windows 7 and the player is embedded on a webpage within a Mozilla Firefox version 25.0.1 browser. The streaming video is the Big Buck Bunny video, each chunk is four seconds long, and has been encoded at 1,300 Kb/s, 850 Kb/s, 500 Kb/s and 250 Kb/s. The alternative videos are encoded using the same parameters.

In the following experiments, we present results which show how different policies perform under different network conditions. For all experiments we use two servers. One server hosts all the videos and the other hosts large files that are downloaded in parallel to generate between one and four competing flows. Dummynet runs on the end hosts.

To evaluate the impact RTTs have on the performance of the policies we keep the RTTs for the competing flow(s) fixed, while running experiments with different RTTs between the client and the streaming service. By running experiments with different number of competing flows, we can capture the bandwidth degradation while experiencing increased competition [13, 15]. In our default experiments we use a single competing flow. Both the video flow and the competing flow experience an RTT of 150ms, as motivated by the average ping latency observed from our university network to the top-million websites (per Alexa.com).

To capture both good and bad network conditions for the encodings and setup above, we evaluate the system for different shared available bandwidths, with 4,000 Kb/s being our default bandwidth. In our experiments, we also take a closer look at the impact of the buffer size and TCP version.

Our instrumented player continually records information about player states and events to a log file. These log files are then processed to generate our results. Unless specifically mentioned, in each experiment the streaming video plays for 90 seconds and each experiment is repeated 20 times.

We also include results using the naive OSMF player without any modifications, which does not do any prefetching of alternative videos, to provide a baseline for comparison. The prefetch policies are all implemented using this as a baseline.

6.2 Playback Initiation of Alternative Videos

Before comparing prefetching policies, we show the potential impact of prefetching on the startup times of alternative videos. For this purpose we compare the startup delays of an implementation that does not perform any prefetching with one that has already prefetched the videos.

In both cases, when the user initiates the playback of an alternative video, the currently streaming video is paused, removed from the playback container, and a new player instance is created and added to the container. The player then initializes parameters using the manifest file (assumed available to the player), and then places a request for the first chunk of the alternative video. In the case the chunk has been prefetched it typically takes our player less than 0.1 seconds to load this chunk from the cache, resulting in a total startup time of 0.6 seconds (with a standard deviation of 0.15 seconds). In contrast, for the case without prefetching the chunk must first be downloaded from the origin server which takes considerable time. For example, in our default scenario (with $B = 4,000Kb/s$, $RTT = 150$ ms, and one competing flow) we had a startup delay of 5.8 ($\sigma = 2.3$) seconds, an order of magnitude larger. With $B = 2,000Kb/s$ the startup delay without prefetching was 10.0 (4.1) seconds, and with $B = 8,000Kb/s$ was 3.6 (1.4) seconds.

While these delays due to downloading the first chunk of the alternative video at first may seem larger than the download of a typical chunk during playback, it is important to remember the state of the system when the client selects to play a different video. In the case the chunk must be retrieved from the origin server, one of two different scenarios typically takes place. First, if there is an ongoing download the client must either make the request over the same connection as the ongoing download (requiring some waiting time before the download can begin) or open up a new parallel connection (in which case the client typically must endure connection establishment, slow start, and simultaneously share the bandwidth with the original connection). Which of these cases takes place is determined by the OSMF download manager, but both result in non-negligible delays as even the lowest-quality chunks typically consist of more than one hundred 1,500 byte packets. While the download typically is faster in the case there is no ongoing download, this case also requires substantial startup delays, especially if having to endure slow start due to timeouts, for example.

6.3 Policy Comparison, Default Scenario

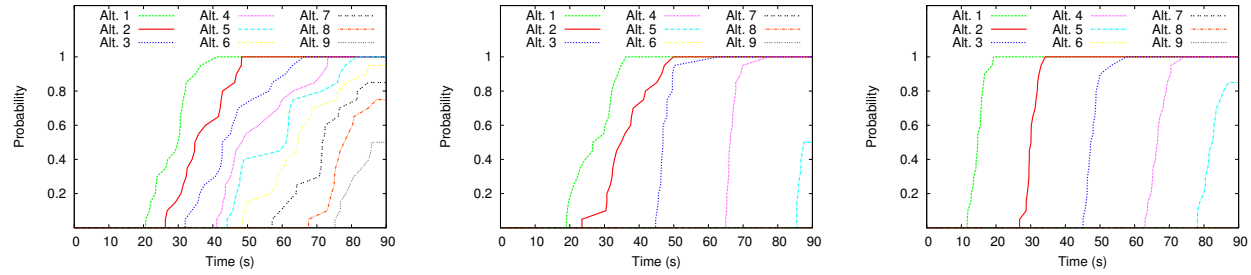
Some of the qualitative differences among the policies are captured by the cumulative distribution function (CDF) of the time stamps at which the different policies have downloaded the first two chunks of the different alternative videos. Figure 4 shows the CDFs of the *lowest quality* policy variants for our default scenario with $B = 4,000Kb/s$, $RTT = 150ms$, $T_{min}/T_{max} = 8/12$, and a single competing flow (also with an RTT of 150ms). The results for the *adaptive quality* variants are qualitatively similar.

Note that while the first CDF with the *best effort* policy and that with the *token-based* policy are similar (as the policies are identical to this point), their characteristics differ substantially for later alternative videos. The *best effort* policy quickly tries to download as many alternative chunks as possible. In contrast, the *token-based* policy paces itself, and downloads a few extra chunks of each video. For this policy a new token was released every 20 seconds, starting at time 0. Most even is the pacing of the *deadline-based* policy. For this policy a deadline was placed every 20 seconds, starting at time 20s. For the default scenario considered here none of the policies experienced playback stalls.

As noted previously, the deadline-driven pacing of the *deadline-based* policy class comes at the cost of a somewhat lower playback quality than for the other policies. Figure 5 quantifies these playback quality differences, and includes results for both the *lowest quality* and *adaptive quality* policy variants. The figure shows the percentage of time the streaming video is playing at different video encodings, as well as the quality of the chunks of the alternative videos downloaded. We note that the *best-effort* and *token-based* policies achieve slightly higher playback quality for the streaming video than the naive player, whereas there are only small differences in the playback quality observed for the *deadline-based* policy class. This difference is due to the optimization framework's need to take into account tighter deadlines to ensure small stall probabilities for the *deadline-based* policies.

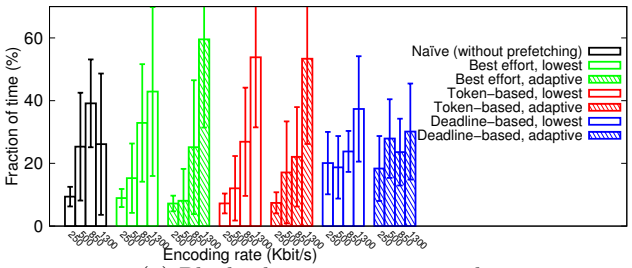
6.4 Impact of Network Conditions

We next take a closer look at the impact of the available bandwidth (B), round-trip time (RTT), and the amount of

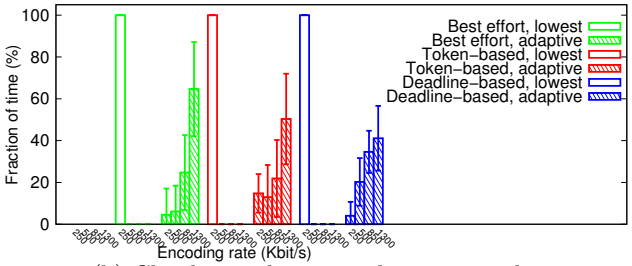


(a) Best effort (b) Token-based (c) Deadline-based

Figure 4: CDF of prefetch completion times for alternative videos under default scenario ($B = 4,000$ Kb/s).



(a) Playback rate streaming video



(b) Chunk encoding rate alternative video

Figure 5: Encoding rates under default scenario.

competing traffic. For this analysis we consider (a) the average stall probability, (b) the playback rate for the streaming videos, and (c) the average encoding rate for the alternative videos downloaded. While we have omitted the number of stalls and stall duration, the Pearson correlation coefficients between the stall probability and these two metrics are 0.94 and 0.71, respectively, for the cases we observe stalls.

Shared available bandwidth: As expected, all policies adapt the streaming quality based on the available bandwidth (Figure 6(b)), and for the *adaptive quality* variant also the chunk quality is adapted (Figure 6(c)).

Referring to Figure 6(b), we can also see that the player performance of all policies (with one competing flow) flattens out somewhere between 6,000 and 8,000 Kb/s shared available bandwidth, and that not much additional performance is gained beyond this point, as the client achieves close to the maximum encoding rate of 1,300 Kb/s.

Naturally, with the *best effort* and *token-based* policies, prefetching will only take place when there is sufficient bandwidth to fill the buffer. In contrast, the *deadline-based* policy prioritizes downloading the initial chunks of the alternative videos by their respective deadlines at the cost of somewhat lower streaming quality (Figure 6(b)).

In general, the stall probabilities (Figure 6(a)) are low or zero, except for the low-bandwidth scenario (with $B = 1,000$ Kb/s). The higher stall probabilities for the case when $B = 1,000$ Kb/s are in large part due to the lack of lower qualities (below 250 Kb/s) causing a smaller error margin, and the coarser chunk granularity at this bandwidth causing a higher penalty of selecting the wrong quality. In this scenario, many of the stalls occur due to clients switching to a higher quality (typically from 250 Kb/s to 500 Kb/s, with all policies having an average playback rate, in this case, of between 259 and 277 Kb/s) and then not getting the next chunk in time, due to a (temporary) degradation of the download speed (e.g., due to the competing flow).

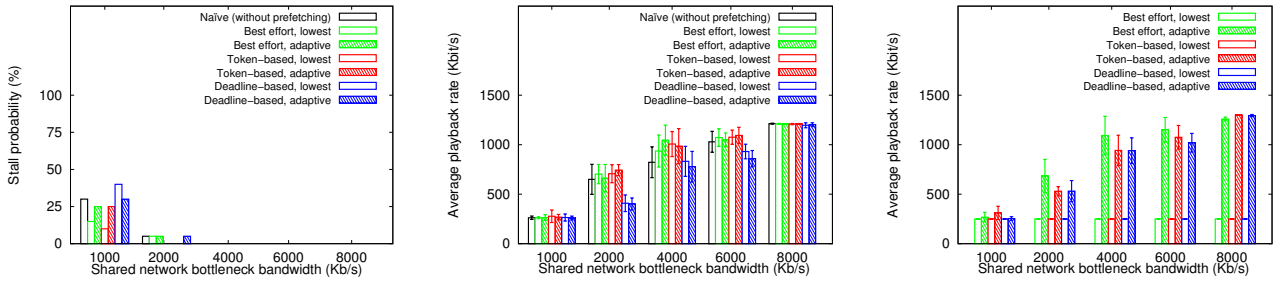
While non-negligible when $B = 1,000$ Kb/s, the stall probabilities of the *best effort* and *token-based* policies are comparatively (slightly) lower than those of the *naive* player. The higher stall probabilities for the *deadline-based* policy (40%) observed for this scenario are due to very tight conditions. For example, to meet the first deadline, more than 28s of video must be downloaded in 16s, and the competing flow hogs much of the shared bandwidth and complicates the bandwidth prediction used in the optimization framework. This shows that the policy is somewhat optimistic for these circumstances, and that prioritizing the alternative videos under conditions with limited and variable available bandwidth can come at the cost of increased stall probabilities.

While we expect that the results could be improved with the help of better prediction algorithms for the available bandwidth, the use of bigger buffers, or by increasing the distance between download deadlines of alternative videos, clearly, giving too high priority to the alternative videos may not be ideal in these constrained (and hard-to-predict) cases. In such cases the simpler *token-based* approach may be preferable.

Finally, for all of the bandwidths, the *adaptive-quality* policy variants are able to download the alternative videos at a much higher quality than the *lowest-quality* variants (Figure 6(c)), without negatively impacting the streaming quality of the streaming video as measured by the stall probabilities (Figure 6(a)) and the playback encoding rate of the streaming video (Figure 6(b)) These results suggest that these relatively simple *adaptive-quality* variants do a good job adjusting to the current network conditions.

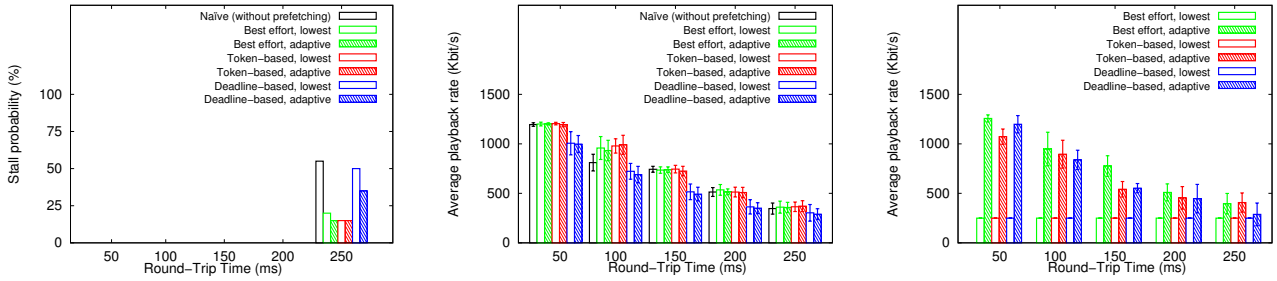
Round-trip times: TCP throughput is greatly impacted by RTTs. We run experiments for different RTTs (between 50ms and 250ms) for our application, while keeping the RTT of the competing flows fixed at 50ms. The results in Figure 7 shows that the policies in general adapt well to the varying network conditions, adjusting the streaming playback rate (Figure 7(b)) to maintain low (typically zero) stall probabilities (Figure 7(a)).

As expected, there is a clear decrease in the playback rate (Figure 7(b)) of the streaming video as the RTT of the video application increases. This trend matches well with the in-



(a) Stall probability (b) Streaming encoding rate (c) Alternative encoding rate

Figure 6: Impact of the shared available bandwidth.



(a) Stall probability (b) Streaming encoding rate (c) Alternative encoding rate

Figure 7: Impact of the round-trip times (RTTs).

verse relationship typically observed between TCP throughput and RTTs. For the case for which we observe stalls, note that the stall probabilities of all policies are similar or lower than for the naive player. This is particularly encouraging for the *adaptive-quality* variants which are able to download alternative videos at a higher quality (Figure 7(c)) than the *lowest-quality* variants without sacrificing playback quality of the streaming video (Figure 7(b)).

Competing flows: The effective bandwidth share of the application decreases with increasing numbers of competing flows traversing the bottleneck, giving similar results (omitted) to those seen with decreasing available bandwidth in Figure 6. Our prefetching policies adapt by adjusting the playback quality, and with the adaptive variants also the encoding of the alternative videos.

Alternative videos downloaded: Another important consideration when selecting which type of policy to use is the number of alternative videos that the policy is able to download (in the case of the *deadline-based* policy, by their individual deadlines). Figure 8 shows this metric for the first 90 seconds of our experiments.

As expected, the *best-effort* policy typically downloads the first two chunks of more alternative videos than the other policies; the differences being particularly apparent when considering the *lowest-quality* variants. For the other policies the *lowest-quality* and *adaptive-quality* variants are able to complete approximately the same number of preloads. Given the higher quality seen with *adaptive-quality* (Figures 6(c) and 7(c)), these results may suggest that the *adaptive-quality* variants are more attractive than the corresponding *lowest-quality* policies.

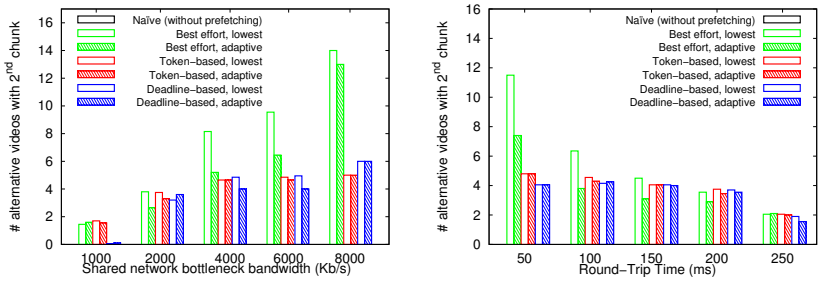
6.5 High-bandwidth Variation Scenarios

We have also evaluated the different policies under real-world commuter traces with widely varying available bandwidths, collected while using different modes of transport in Oslo [25]. The *ferry trace* has an average bandwidth $\overline{B}$ of 2,026 Kb/s and a standard deviation (σ) of 1,020. The *bus trace* has $\overline{B}$ = 2,031 Kb/s and σ = 432; the *metro trace* has $\overline{B}$ = 1,014 Kb/s and σ = 470; and finally the *tram trace* has $\overline{B}$ = 1,082 Kb/s and σ = 286.

Among these traces, the ferry trace is perhaps the most extreme as the available bandwidth varies widely depending on the distance to land. The variation is high also for the metro trace. For these two traces, all policies (including the naive player) have stall probabilities above 75%. The *deadline-based* policies also suffer from significant stall probabilities (around 50%) for the tram trace. For all other policies and scenarios we observe only negligible stall probabilities. In general, as for the previous scenarios, both the *best-effort* and *token-based* policies are able to achieve similar or better performance than the naive player, while also prefetching alternative videos for instantaneous playback. This is examplified by the playback encoding rates shown in Figure 9.

6.6 Buffer Size

HAS clients make use of a playback buffer to accommodate for network jitter and short-term variations in the available bandwidth. The typical buffer size varies between applications, playback device, and in some cases also on the network in which the application and device are deployed. We have validated our conclusions for T_{min}/T_{max} = 4/6, 8/12, 8/16, 12/24, 12/30 (seconds). Note that as $T_{max} - T_{min}$ increases in these configurations, there will be fewer off-periods, but of greater duration (equal to 2, 4, 8, 12, and 18 seconds, respectively). While we observed much fever stalls for bigger buffer sizes, the relative performance across policies and against the naive player are consistent with the results presented earlier in the paper. (Figures omitted.)



(a) Bandwidth (b) RTT

Figure 8: Number of alternative videos with at least two chunks.

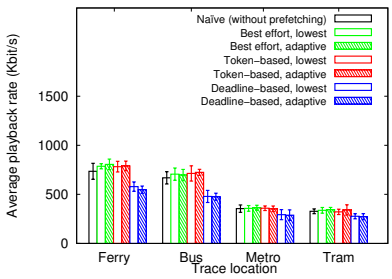


Figure 9: Streaming encoding rate in trace-driven scenarios.

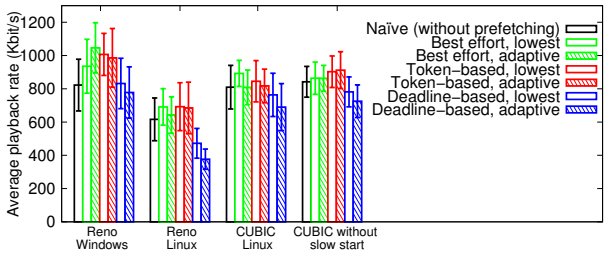


Figure 10: TCP version comparison.

6.7 Slow Start and TCP Configurations

The performance under competing traffic also varies between TCP versions. The impact of off periods is particularly apparent in cases when slow start is invoked after the off periods, or the congestion window, due to other reasons, takes more time to reach its fair-share bandwidth.

To understand the impact of TCP slow start and congestion control in general, we compared the performance of four alternative TCP implementations: (i) Reno on Windows, (ii) Reno on Linux, (iii) CUBIC on Linux, and (iv) a modified version of CUBIC on Linux that retains its window size during off periods. We note that the three first versions by default reduce the number of segments that the server can send at a time whenever the flow has remained inactive for duration that is longer than a timeout threshold.

While the performance improvements achieved by the *best-effort* and *token-based* policies are largest with the two Reno implementations, our conclusions are consistent across TCP implementations. Both these policies consistently give the same or better performance than the naive player. For example, Figure 10 shows the average playback rate for the default scenario (with $B = 4,000$ Kb/s). Here, the *best-effort* policy achieve average improvements in the playback rate (relative to the naive player) of 20.5% and 8.1% with the two Reno implementations, respectively, 5.1% with CUBIC, and 2.6% using CUBIC without slow start. Not surprisingly, the smallest improvements are seen when using CUBIC without slow start. These observations are expected as CUBIC ramps up faster than Reno, but not as quickly as the CUBIC implementation that avoids slow start after inactive periods. The prefetch policies, on the other hand, effectively avoid off periods altogether.

7. RELATED WORK

Playback stalls and frequent quality switches have been shown to have the most significant impact on user satisfaction and video abandonment during playback [10, 11]. The HAS player adaptation algorithm plays an important role here [5]. Inaccuracies in the rate estimations [17] and interaction with other competing flows [3, 13] have been found to negatively impact the performance of many HAS players. TCP is often part of the problem, as clients may be forced to resume the download from slow start after an idle period. Although observed as a problem already for traditional web workloads [22], this feature remains the default in several TCP implementations.

For long duration content, others have suggested that rate estimates are only important for the initial transient period, when building up a playback buffer, and that buffer-based policies that track changes in buffer size can be used in steady-state [9, 14]. Rather than looking at the steady-state period of such large-buffer and long-duration video systems we consider preloading during the transient period, when users are more likely to switch to an alternative video.

In order to be fair to multiple competing clients, server-based traffic shaping has been proposed [4]. Other solutions have relied on optimized chunk scheduling, bandwidth estimation, and quality selection that ensures fair and stable playback characteristics [15].

Prefetching has long been used to improve Web performance [23], but has also been applied in the context of HAS-aware proxies that preemptively try to download chunks that the clients are likely to download next [18]. Others have observed that some players may re-download chunks that have already been downloaded, at a higher quality [27]. In this paper, we show how careful prefetching can be used to improve HAS performance, by stabilizing the client's bandwidth share and simultaneously allowing preloading of alternative videos, but do not consider policies that use the extra bandwidth to re-download chunks at higher quality. Combining these ideas could be interesting future work.

Many recommendation algorithms and systems have been proposed, including to determine which video a user is likely to want to view next [7, 8, 12, 28]. In addition to client-side recommendations, video recommendations have also been used to improve cache efficiency [20].

While both the prefetching framework and the implementation to allow quick preloading of alternative videos are novel, we are not the first to combine multiple videos into a seamless playback experience. For example, Johansen et al. [16] use a modified player to combine video snippets based on tagged videos and user search queries into a single personalized video stream. Krishnamoorthi et al. [19] implement an OSMF-based client that is able to provide seamless playback of interactive branched video. The DASH specification also allows ad insertion into live and VoD streams [1].

8. CONCLUSIONS

This paper presents the design, implementation, and evaluation of a HAS-based framework that uses careful prefetching and buffer management to preload the initial chunks of alternative videos. We show that our design and implementation allows prefetching and instantaneous playback of prefetched alternative videos without any playback quality degradation of the currently streamed video. To capture a wide range of objectives, three classes of prefetching policies are designed and evaluated.

The design, characteristics, and performance of the three policy classes (i.e., *best-effort*, *token-based*, and *deadline-based*), and their variations (i.e., *lowest-quality* and *adaptive-quality*), are compared against each other and the naive player on top of which they are all implemented. As the policy classes are designed with different objectives, these comparisons provide insights into desirable policy choices for providers with different objectives. Perhaps most encouraging is the performance of the *token-based, adaptive-quality* policy. This policy (i) allows the download of alternative videos to be relatively well paced for a wide range of network conditions, (ii) nicely adapts the quality of the prefetched chunks based on the current conditions, and (iii) provides as good, and often better, playback quality as the naive player, as exemplified by similar stall probabilities and better playback encoding rate. The *best-effort, adaptive-quality* policy performs similarly except does not pace alternative video downloads. Finally, the *deadline-based* policies trade some of the playback quality of the streaming video for prefetch deadlines and more even pacing.

Much of the improvements in bandwidth utilization and streaming performance shown in this paper are achieved by leveraging the off periods typically observed with HAS players. In this paper we have leveraged rate estimates to decide when and at what quality to download chunks of the alternative videos. Recent work has suggested that in steady state buffer-based policies (rather than rate-based policies) should be used to adapt the playback quality [14]. Future work will consider such generalizations.

9. REFERENCES

[1] 3gpp tr 26.938 version 12.0.0 release 12, 2014.

[2] Sandvine global internet phenomena report- 2h2014-technical report, 2014.

[3] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. What happens when HTTP adaptive streaming players compete for bandwidth? In *Proc. ACM NOSSDAV*, 2012.

[4] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen. Server-based traffic shaping for stabilizing oscillating adaptive streaming players. In *Proc. ACM NOSSDAV*, 2013.

[5] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proc. ACM MMsys*, 2011.

[6] L. Chen, Y. Zhou, and D. Chiu. A study of user behavior in online vod services. *Computer Communications*, 2014.

[7] P. Cui, Z. Wang, and Z. Su. What videos are similar with you?: Learning a common attributed representation for video recommendation. In *Proc. ACM Multimedia*, 2014.

[8] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and D. Sampath. The YouTube video recommendation system. In *Proc. ACM RecSys*, 2010.

[9] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. ELASTIC: A client-side controller for dynamic adaptive streaming over HTTP (DASH). In *Proc. IEEE Packet Video Workshop*, 2013.

[10] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. In *Proc. ACM SIGCOMM*, 2011.

[11] T. Hossfeld, M. Seufert, C. Sieber, T. Zinner, and P. Tran-Gia. Identifying QoE optimal adaptation of HTTP adaptive streaming based on subjective studies. *Computer Networks*, 2015.

[12] Q. Huang, B. Chen, J. Wang, and T. Mei. Personalized video recommendation through graph propagation. *ACM TOMM*, 2014.

[13] T. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, timid, and unstable: Picking a video streaming rate is hard. In *Proc. ACM IMC*, 2012.

[14] T. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proc. ACM SIGCOMM*, 2014.

[15] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive. In *Proc. ACM CoNEXT*, 2012.

[16] D. Johansen, P. Halvorsen, H. Johansen, H. Riiser, C. Gurrin, B. Olstad, C. Griwodz, Å. Kvalnes, J. Hurley, and T. Kupka. Search-based composition, streaming and playback of video archive content. *MTA*, 2012.

[17] Z. Kelvin, J. Erman, V. Gopalakrishnan, E. Halepovic, R. Jana, X. Jin, J. Rexford, and R. Sinha. Can accurate predictions improve video streaming in cellular networks? In *Proc. ACM HotMobile*, 2015.

[18] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. Helping hand or hidden hurdle: Proxy-assisted http-based adaptive streaming performance. In *Proc. IEEE MASCOTS*, 2013.

[19] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. Quality-adaptive prefetching for interactive branched video using http-based adaptive streaming. In *Proc. ACM Multimedia*, 2014.

[20] D. Krishnappa, M. Zink, C. Griwodz, and P. Halvorsen. Cache-centric video recommendation: An approach to improve the efficiency of YouTube caches. In *Proc. ACM MMSys*, 2013.

[21] T. Kupka, P. Halvorsen, and C. Griwodz. Performance of on-off traffic stremming from live adaptive segmented HTTP video streaming. In *Proc. IEEE LCN*, 2012.

[22] V. Padmanabhan and R. Katz. Tcp fast start: A technique for speeding up web transfers. In *Proc. IEEE Globecom*, 1998.

[23] V. Padmanabhan and J. Mogul. Using predictive prefetching to improve world wide web latency. *ACM CCR*, 1996.

[24] A. Rao, A. Legout, Y. Lim, D. Towsley, C. Barakat, and W. Dabbous. Network characteristics of video streaming traffic. In *Proc. ACM CoNEXT*, 2011.

[25] H. Riiser, H. S. Bergsaker, P. Vigmostad, P. Halvorsen, and C. Griwodz. A comparison of quality scheduling in commercial adaptive HTTP streaming solutions on a 3G network. In *Proc. Workshop on Mobile Video*, 2012.

[26] L. Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *ACM CCR*, 1997.

[27] C. Sieber, A. Blenk, M. Hinteregger, and W. Kellerer. The cost of aggressive HTTP adaptive streaming: Quantifying YouTube's redundant traffic. In *Proc. IFIP/IEEE IM*, 2015.

[28] A. Xavier. Mining large streams of user data for personalized recommendations. *ACM SIGKDD Explor. Newsl.*, 2012.