Helping Hand or Hidden Hurdle: Proxy-assisted HTTP-based Adaptive Streaming 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*—HTTP-based Adaptive Streaming (HAS) has become a widely-used video delivery technology. Use of HTTP enables relatively easy firewall/NAT traversal and content caching. While caching is an important aspect of HAS, there is not much public research on the performance impact proxies and their policies have on HAS. In this paper we build an experimental framework using open source Squid proxies and the most recent Open Source Media Framework (OSMF). A range of content-aware policies can be implemented in the proxies and tested, while the player software can be instrumented to measure performance as seen at the client. Using this framework, the paper makes three main contributions. First, we present a scenario-based performance evaluation of the latest version of the OSMF player. Second, we quantify the benefits using different proxy-assisted solutions, including basic best effort policies and more advanced content quality aware prefetching policies. Finally, we present and evaluate a cooperative framework in which clients and proxies share information to improve performance. In general, the bottleneck location and network conditions play central roles in which policy choices are most advantageous, as they significantly impact the relative performance differences between policy classes. We conclude that careful design and policy selection is important when trying to enhance HAS performance using proxy assistance.**

***Keywords*—Proxy-assisted; HTTP-based adaptive streaming; Prefetching; OSMF; Experimental evaluation**

I. Introduction

For the last two decades video streaming has been expected to be the next killer application. Today, popular services such as YouTube and Netflix are offering large catalogues of user generated and professional videos, respectively. Video streaming has finally become mainstream, playing an increasingly important role in our everyday lives. It was recently reported that streaming media is responsible for close to 40% of the downstream traffic in production networks, and it is predicted that streaming media will account for two-thirds of the traffic in the coming years [1].

While much of the early video streaming work focused on UDP-based solutions, today the majority of streaming traffic is delivered using HTTP over TCP. Use of HTTP enables both simple firewall traversal and effective content caching. These two factors provide immense benefits, as HTTP can reach almost any user, and content can easily be replicated at locations closer to the client.

With basic HTTP-based streaming, the video encoding that is delivered to a client is fixed at the time of the client request. To increase the quality of service and to better utilize the available network bandwidth, HTTP-based Adaptive Streaming (HAS) is being increasingly adopted by content providers [5]. With HAS, the video is encoded into different qualities and clients can at each point in time adaptively choose the most suitable encoding based on the current conditions.

While content caching plays an important role in the motivation for HTTP-based streaming, only very limited work considers the impact of using proxy assistance for HAS.

In this paper we present a performance evaluation of proxy-assisted HAS. Our experimental framework is designed using a popular open source proxy and a popular open source media player. Careful instrumentation allows us to measure the actual performance seen at the client, while controlling the network conditions and protocol parameters. We evaluate a range of different policy classes that may be employed in proxy-assisted systems and provide insights to the effectiveness of different policies and their performance tradeoffs.

While the general proxy policy classes considered in this paper are applicable for any chunk-based HAS protocol, in our evaluation we use Adobe's Open Source Media Framework (OSMF), together with the accompanying (open source) Strobe Media Playback (SMP) media player. In addition to capturing the *actual* performance seen on the client, using an open source player also allows us to easily consider the impact buffer sizing at the client has on proxy solutions, and avoids other issues that may come with using proprietary players (e.g., having to decipher encrypted manifest files [3]). Our proxy-assisted policies are all implemented using open source Squid proxies.

Leveraging our experimental framework and instrumentation, the paper makes three major contributions. First, we carry out an experimental evaluation of the latest version of SMP. Our results illuminate several important aspects of SMP performance, in particular relating to player buffer sizing.

Second, we analyze the player's performance in a proxy-assisted environment, considering a wide range of proxy policies including both basic baseline policies (such as a best-effort proxy that only caches chunks that it has served other clients in the past), and more advanced content-aware prefetching policies in which the proxy prefetches and caches chunks likely to be requested in the future (such as the chunk following that last requested by a client, at the same quality level).

Performance with the alternative proxy policies is found to be quite sensitive to the network conditions. If the bandwidth

To appear in *Proc. IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, San Francisco, Aug. 2013. The final/official version will appear in the conference proceedings and the IEEE Xplore Digital Library.

bottleneck is between the client and the proxy, we find that a relatively simple proxy solution can provide most of the potential benefits, but these benefits are modest. However, if the bottleneck is between the proxy and the server, caching at the proxy can yield a substantial benefit, but the limited bandwidth between the proxy and server must be more carefully utilized. There is traffic overhead associated with prefetching, and aggressive prefetching can therefore result in performance degradation. For example, a client that sees high hit rates at the proxy can be lured into selecting a higher quality than the proxy is able to serve if the future chunks requested by the client are not available at the proxy.

Third, we present a novel collaborative client-proxy framework in which clients and proxies share information with the goal of improving both the cache hit rate at the proxy, and the video quality at the client. Our performance evaluation results indicate that this goal can be achieved, yielding substantial benefits in some cases. Our buffer-size experiments also suggest that advanced policies are particularly attractive when assisting wireless clients with smaller buffers, for which unnecessary bandwidth usage may be more costly.

The remainder of this paper is organized as follows. Section II presents background on HAS and SMP. Section III describes our experimental methodology, performance metrics, and instrumentation. Section IV presents a scenario-based evaluation of SMP performance. Section V describes our performance evaluation of proxy-assisted HAS. Section VI discusses related work and Section VII concludes the paper.

II. Background

A. HTTP-based Adaptive Streaming

With HTTP-based streaming, either the quality level is statically chosen when the video is requested, or adaptively varied during playback. We refer to the second approach as HTTP-based Adaptive Streaming (HAS).

Typically HAS protocols use some variation of either chunk requests or range requests. With a purely chunk-based approach, used by OSMF and Apple's HTTP Live Streaming (HLS), for example, each encoding of each video is split into many smaller chunks. Each chunk is given a unique URL, which the clients can learn about from the video's manifest file. At the other end of the spectrum, e.g. used by Netflix, each encoding of a video is stored in a single file and clients use standard HTTP range requests to download ranges of bytes from these files. Hybrid approaches are also possible. For example, Microsoft Smooth Stream (MSS) uses a unique URL in the manifest file to specify each unique chunk, but then applies a server-side API to map chunk URLs into file ranges.

In this paper we consider chunk-based adaptive streaming, as used by the Open Source Media Framework (OSMF). OSMF is open source and allows us to modify and instrument the player used in our experiments to directly measure the performance seen at the player. OSMF is a widely used design framework that includes a library of the necessary components to create an online media player. It also comes with the Strobe Media Playback (SMP), a fully pre-packaged media player for deploying OSMF. Other open source players include a VLC plugin [20] and GPAC [12]. For simplicty, here we use OSMF and SMP interchangeably.

B. SMP overview

1) Buffer management: Similar to other HAS protocols, including previous versions of OSMF, the most recent version of the SMP media player (version 2.0) controls content retrieval based on (i) its buffer occupancy, and (ii) the estimated available bandwidth. With OSMF's HTTP Dynamic Streaming (HDS) protocol, each encoding of a video is divided into smaller fragments that typically have a 2-5 seconds playback duration. The player issues requests for these fragments successively and the server delivers the fragments statelessly to the client. The protocol uses two buffer thresholds: the minimum (T_{min}^{buf}) and the maximum (T_{max}^{buf}) buffer time threshold.

During normal operation, the player tries to maintain at least T_{min}^{buf} (in seconds) buffer occupancy at all times, and begins playback only when there is at least T_{min}^{buf} seconds of data in the buffer. Generally, the player downloads until T_{max}^{buf} seconds of video is buffered, at which point it stops download until the buffer occupancy is less than the lower threshold T_{min}^{buf} again, at which point download is resumed. In this way, the player's buffer occupancy oscillates between these two threshold values.

2) Rate adaptation: Rate adaptation requires estimation of the available bandwidth (i.e., achievable download rate). This is accomplished using a weighted (rolling) average over the two most recently downloaded fragments. After download of a fragment, the player calculates the average download rate for that fragment and updates its estimated current download rate.

When downloading the next fragment, the most recent estimates of the available bandwidth are used to identify a list of candidate encoding rates that are sustainable for that bandwidth. For each candidate encoding, a reliability value is calculated based on the number of switching decisions of different types that have been made within a fragment-count-based window and the number of stalls encountered at these encodings. Among the encodings with a reliability value higher than a minimum threshold, the player picks the encoding with the highest reliability value. If there are no such encodings, then the lowest quality is chosen.

In addition to the above core mechanism, there are additional *emergency-based* or *threshold-based* rules that impact the player's decisions. These high-priority rules are designed to ensure that the player is not too aggressive/conservative, and include rules that prevent choosing a lower quality when the buffer is sufficiently full (threshold-based rule), and event-triggered (emergency) rules that keep track of empty buffer occurrences, dropped frames, and quality up-switches.

III. Methodology

This paper evaluates HAS performance as seen at the client player, under different player and proxy policies, system configurations, and/or architectures. In contrast to prior works, which infer the player behavior based on network measurements [3], [6], [24], we evaluate the *actual* performance seen at the player. By instrumenting the player we can avoid any potential misinterpretations due to TCP buffer management and/or other hidden factors over which the player may have limited control.

A. Experimental setup

We implement and run a server, proxy, and client on three separate machines, all connected over the same 100Mbit/s LAN. To emulate a wide range of network characteristics, including wide-area content retrieval, we use dummynet [25] to control the network conditions between the client and proxy, as well as between the proxy and server.

The server machine is running a Flash Media Server and is hosting a 10 minute video available at four encoding rates: 1300 Kbit/s, 850Kbit/s, 500Kbit/s and 250Kbit/s, with key frames set four seconds apart. The key frame distance is vital in terms of the player's quality adaptation as rate switching can take place only at key frames. We have chosen encoding rates and key frame separation to keep the processing at the client and server modest. This allows us to focus on the network and proxy-related aspects. Levkov [17] provides a detailed discussion on the tradeoffs in picking a good key frame distance. Finally, we use open source Squid proxies to evaluate a range of different proxy implementations/policies.

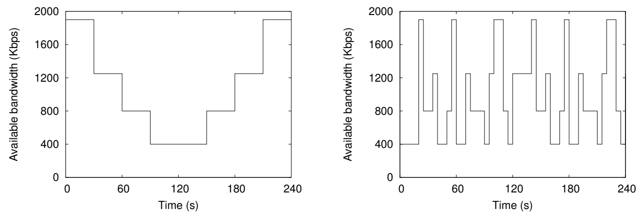
B. Player instrumentation and metrics

For the purpose of player evaluation, we instrument the SMP media player to output important events and parameter information to log files for later analysis. This allowed us to capture the current state of the player at almost every player event, including when a new fragment is requested, when a decision is made to change the quality level, when observing server responses to rate-switch decisions, as well as at the times of other more subtle client-side events. For each such measurement, perhaps the most important information collected is the current buffer occupancy (defined as the amount of video content that is in the player buffer, measured in seconds) and video encoding rate.

In order to characterize the player performance we used the log files from our instrumented client player to calculate a set of metrics that capture both differences in performance and general adaptation characteristics.

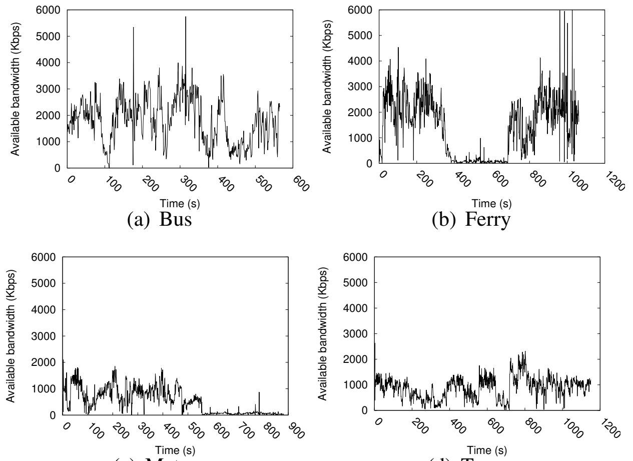
- **Quality level:** The fraction of time the player spends at each quality level during playback.
- **Quality switches:** The number of quality level changes that the player makes per minute.
- **Stall time:** The total time the player is stalled.
- **Buffer interruptions:** The number of times per minute, regardless of how short, that the player is interrupted during playback, due to lack of data in its buffer.

Unless stated otherwise, throughout the paper we present average values (with 95% confidence intervals) calculated over at least 10 full experiments. We also consider proxy-specific metrics, such as the *hit rate*, defined as the fraction of the requested fragments that are found in the proxy cache, and the corresponding *bandwidth savings* due to these fragments not having to be obtained from the server. The caching decisions made when serving one client can impact subsequent clients, and for this reason, in some cases, we show performance conditioned on the number of previous clients that has viewed the same content as the current client. Finally, to provide additional insights, we occasionally refer to the raw event logs.



(a) Slow variations (b) Fast variations

Fig. 1. Synthetic baseline traces for the available bandwidth.



(a) Bus (b) Ferry (c) Metro (d) Tram

Fig. 2. Real-world traces of available bandwidth.

C. Scenario-based evaluation

In our experiments we use both *synthetic bandwidth traces* that capture artificial baseline bandwidth scenarios, as well as *real-world bandwidth traces* that capture the bandwidth dynamics in four example environments. In all scenarios, we use dummynet to control (i) available bandwidth, (ii) loss rate, and (iii) round-trip time (RTT). This control can be exercised so as to emulate the characteristics of a network bottleneck between the client and proxy, and between the proxy and server. While this paper only shows a representative set of example results, in which we typically change one factor at a time, while controlling the others, we note that we typically used full factorial experiments.

1) Synthetic traces: To study the impact of the available bandwidth and how quickly it changes, we use artificial baseline scenarios. Figure 1 illustrates two example scenarios, with slowly and rapidly varying available bandwidth, respectively. We also use a third scenario with static available bandwidth. The average bandwidth (725 Kbit/s) is the same in all three scenarios, and the total time spent at each bandwidth availability level is the same in the scenarios with slow and fast bandwidth variations. In fact, for these scenarios we picked the average time spent at each level to be the same for all levels, such that in the ideal case (in which the player always picks exactly the quality level that is the highest possible based on the current available bandwidth) the player would spend exactly the same amount of time at each quality level. Of course, this is not expected to be achieved in practice.

2) Real-world traces: The real-world bandwidth traces were collected and shared by Riiser et al. [24]. The traces capture the available bandwidth seen in a 3G UMTS network in different parts of Oslo. Each trace represents a different mode of transport (bus, ferry, metro or tram) and allows us to provide insights into how the player performance is influenced by different geographical and physical factors. Figure 2 shows the available bandwidth under the different scenarios.

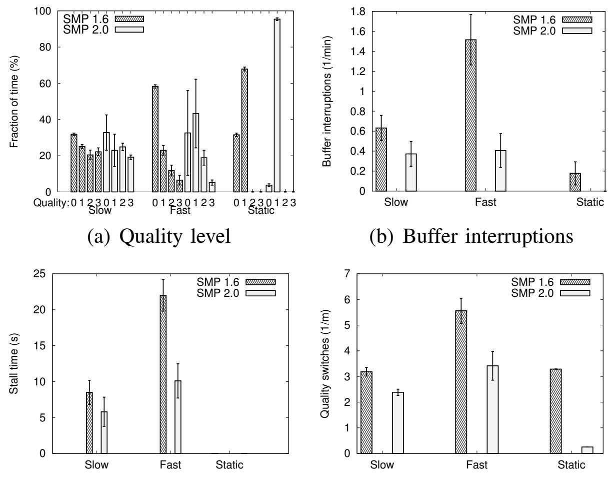
Each of the four traces exposes the player to distinctly different conditions and variations in the available bandwidth. The high available bandwidth in the bus scenario often is well above our highest encoding rate. The ferry route is interesting in that it illustrates a scenario in which there are highly different conditions, depending on the client location. When the client initially is on land the available bandwidth is high, the observed bandwidth is low while on the ferry, followed by high availability when back on land. These changes are clearly visible in the bandwidth plot. It is interesting to note here that the available bandwidth on the ferry is lower than our lowest available encoding rate while the bandwidth observed on land is very good. The metro trace captures the available bandwidth as the train initially is above ground and then enters a tunnel, as reflected by the abrupt change from an available bandwidth in the 500-1400 Kbit/s range, to a period with poor bandwidth availability. Finally, the tram trace shows relatively constantly time-varying bandwidth availability. It is important to note that the limited bandwidth availability in both the metro and tram traces may suggest that these clients should not be expected to view the video at our highest encoding rates.

3) Loss rate and RTT: To capture the impact of loss rates and round-trip times, for each of the above scenarios, we use dummynet to analyze five sub-cases of each of the above scenarios. In the basic case of a single client-server connection, these can be summarized as follows: (i) no additional delay or packet losses, (ii) no delay and 2.5% packet loss rate, (iii) no delay and 7.5% packet losses, (iv) 50ms RTT and no packet losses, and (v) 200ms RTT and no packet losses. Note that the first case corresponds to the ideal loss/delay conditions, while other cases may be more realistic. The use of 2.5% and 7.5% packet loss rates are selected as they may be representative of acceptable packet loss rates in wireless environments. The RTT values are based on intra and cross-continental traffic. To put the RTT in perspective we performed traceroute measurements to the top million websites according to `alexa.com` on June 12, 2012. These measurements suggest that a client on the Linköping University campus would see an average RTT for these sites of 177.4ms (and an average hop count of 17.74).

IV. Player characterization

Before considering the impact proxies have on the player performance, it is important to first build an understanding of the player characteristics and baseline performance. As proxies potentially may cause additional variability in download rates (e.g., due to the difference in cache hits/misses) we pay particular attention to the impact of buffer-size dynamics.

To provide insights into the default performance of the SMP 2.0 player and further motivate its use in experimental studies, we first present results using SMP 1.6 and SMP 2.0. Figure 3 shows the player performance for the two players



(a) Quality level (b) Buffer interruptions

(c) Stall times (d) Quality switches

Fig. 3. Player performance comparison of SMP 1.6 and 2.0 under the three synthetic baseline scenarios.

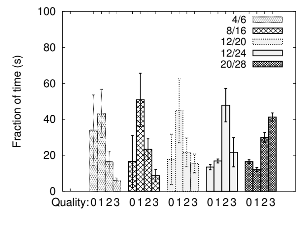
under each of the three synthetic workloads, each capturing a different degree of bandwidth variation. We observe that SMP 2.0 shows substantial improvements over the old SMP 1.6. In fact, it is noteworthy that for all three scenarios the new version outperforms the old player according to all four metrics. For example, in the static bandwidth case, SMP 2.0 quickly finds the maximum sustainable video quality and maintains that quality for the rest of the session. More than 95% of the time is spent in this state (Figure 3(a)). While omitted due to space constraints, similar results have been observed for our other scenarios, loss rates, and end-to-end delays.

Relative to the proprietary Netflix and MSS players, the SMP media player has a much smaller buffer, hinting that it is designed for shorter video playback. While the small buffer comfortably can accommodate for some fluctuations in the available bandwidth, during long periods of high bandwidth variability, the new player may still yield unsatisfactory performance. In general, it appears that SMP is dimensioned based on (relatively) static bandwidth scenarios, for which it is able to quickly rise to the highest possible encoding and maintain that quality. The smaller buffer footprint may also have advantages for legacy devices.

To obtain a better understanding of the impact of buffer-size dynamics, we modified the player so that it could be run with different buffer size configurations.[1] In addition to the default buffer configuration $T^{buf}_{min}/T^{buf}_{max} = 4/6$, we also used: 8/16, 12/20, 12/24, 20/28.

Figure 4 shows the observed video quality and stall times for the fast varying bandwidth scenario with a RTT of 50ms. The figure clearly shows that increasing the buffer size has positive effects on both the video quality being played and the stall times. For example, we are able to play with zero stalls for buffer sizes as small as 12/24. With larger buffer sizes (e.g., at 20/28), aided by long-term buffering effects, we are even able to play at the highest quality for more than 25% of time, despite the available bandwidth only exceeding the highest encoding rate for 25% of the time.

[1]To allow proper buffer management both buffer management and rate adaptation mechanisms had to be modified to account for the new buffer thresholds. In general, we tried to keep these modifications to a minimum.


(a) Quality

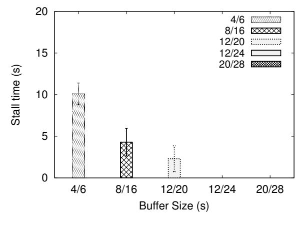
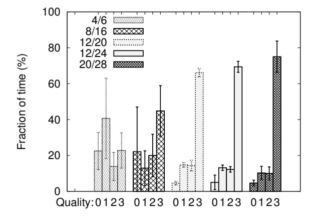
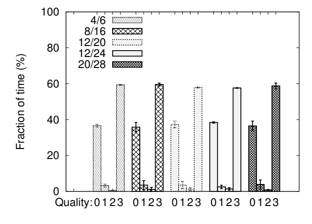
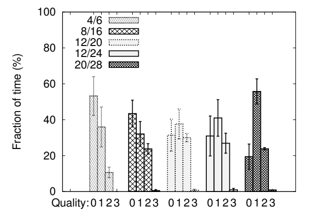

(b) Stall time

Fig. 4. Performance impact of buffer sizes using the fast varying synthetic bandwith trace.


(a) Bus


(b) Ferry


(c) Metro

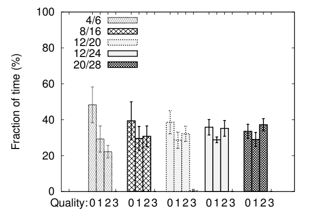
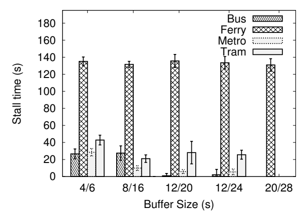

(d) Tram

Fig. 5. Video quality under real-world scenarios for different buffer sizes.


(a) Stall times

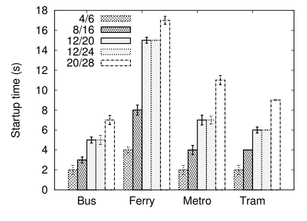

(b) Startup delays

Fig. 6. Stall times and startup delays under real-world scenarios for different buffer sizes.

Figures 5 and 6(a) show the observed video quality and stall times for different buffer sizes, when running experiments using the real-world traces to emulate the available network bandwidth. For three of the real-world scenarios, we observe highly positive effects using larger buffer values. For example, the 20/28 configuration is able to eliminate stall times and allow high overall quality encodings for the bus, metro, and tram traces. It should be noted that for the metro and tram traces, this is achieved despite the available bandwidth not being sufficient to achieve the highest quality level for much of the trace duration. For these scenarios, Figures 4(a) and 6 suggest that there may be an inverse relationship between the buffer size and stall times, with much of the improvements related to T_{min}^{buf}, which determines when the player resumes/starts playback. In contrast, in the ferry scenario, the larger buffer sizes provide little or no benefit. This is explained by the extreme workloads conditions during this trace: either very high bandwidth or very low bandwidth. For completeness, Figure 6(b) shows the corresponding startup delays.

V. Longitudinal proxy performance

This section describes a performance evaluation aimed at providing insights into the impact proxies and their policies may have on HAS performance. While caches and middle boxes already are widely deployed across the Internet, their impact on chunk-based content delivery, as exemplified here by HAS, can also give hints on the design of future content-centric architectures [15]. For this reason, we consider a wide range of policies, including both policies that can be implemented in existing standard proxies, as well as more advanced policies that take into account the structure of the content itself. For our experiments we implemented and tested our policies in Squid (version 2.7-stable9), a popular open source cache that allows us to create our own caching and prefetching rules.

A. Baseline policies

Before designing advanced proxy-assisted policies, it is important to understand the performance impact of proxies and their policies under the most basic conditions. We first define a set of baseline policies which captures the basic performance tradeoffs seen in existing systems, and give some insights to how much room for improvement there may be.

- **Empty cache**: To capture the performance penalty associated with cache misses, we consider a policy in which the cache is cleared for each new client. This would correspond to the performance seen by the first requesting client, including clients that are requesting contents from the long tail of one-timers [13], [19]. In this case, every fragment that the player requests will be fetched from the server.
- **Full cache (all versions):** To capture the performance under ideal proxy conditions we also include a policy in which the cache has been preloaded with all versions of every fragment. In this case, there will be no cache misses and the client can always be served from the cache, resulting in minimum transfer times.
- **Best effort:** To capture the performance of a standard proxy cache, that has seen a limited number of previous client requests, we use a simple policy that has cached every fragment that previously has been requested. In our experiments we start with an empty cache, pick a large cache size such as to avoid any cache replacement, and run the experiments for a sequence of clients.

While the first two policies correspond to the case of minimal and maximum cache hit rates, respectively, the third best-effort policy, captures the more realistic case of intermediate cache hit rates. On average, the cache hits will typically increase as more clients will have requested the same contents in the past, potentially reducing the transfer delays and increasing the TCP throughput for these clients.

For the best-effort policy we initialize all our experiments with an empty proxy cache, and allow one client to access the content at a time. This captures the performance observed by the n^{th} client arriving to the system, assuming that there have been $(n-1)$ previous clients accessing the content and the cache does not remove any of the content that has been accessed in the past. In practice, a cache replacement policy, such as Least Recently Used (LRU), would typically be used. To interpret the results in this context, we note with the LRU policy there exists a time threshold (unique for each time instance) such that all fragments that were requested after this time are in the cache, and none of the other fragments are in the cache. Based on this observation, we can interpret the above abstraction as a cache operating under the LRU policy for which the current time threshold (of what is in the cache) is such that the content requested by the last $(n-1)$ clients are in the cache, but no other fragments. Having said this, it is important to note that this typically would result in a conservative estimate, as the average cache hit rate that a LRU system in steady state likely would require slightly less storage. In practice, such a system would likely see more frequent (and similar) accesses to low quality contents, which may be frequently accessed during the transient time period during which the cache is being filled.

B. Quality and content-aware prefetching

We next describe three basic content-aware prefetching policies that take into account the content structure, as well as the quality level that is currently being requested.

- **1-ahead:** In the most basic prefetching policy, the proxy fetches the next fragment with the same quality as the fragment that is currently being downloaded. When the client has fairly stable bandwidth this policy should yield higher cache hit rates. The policy may waste bandwidth whenever the client switches quality.
- **n-ahead:** A somewhat more aggressive policy is to prefetch up to n fragments ahead of the currently requested fragment, but still only prefetching fragments of the same quality level as the currently requested fragment. Under scenarios with many quality switches this policy can result in much wasted bandwidth between the proxy and server.
- **Priority-based:** To take advantage of the case when the cache already has the content that may be the most natural to prefetch next, we present a priority-based policy that (i) only prefetches one fragment ahead, but (ii) takes into account both the current cache content and the status of the client (as seen from the perspective of the proxy) when selecting what quality this fragment should have. In the case that the client last switched to a higher encoding *and* it is not the first time that the client is requesting fragments of this video quality (i.e., at some earlier time it has switched down from this quality), we use the following priority: (i) current quality, (ii) one quality level below, (iii) one quality level above, and (iv) no prefetching. In all other cases (i.e., the client last switched to a lower quality or has never been at this quality level before), we use the following priority: (i) current quality, (ii) one quality level above, (iii) one quality level below, and (iv) no prefetching.

When interpreting the priority-based policy, it is important to note that the player always starts at the lowest level and initially is trying to ramp up to as high a quality level as possible. After it has reached a sustainable quality level it would typically oscillate between two neighboring quality levels. In contrast to the two first policies, the priority-based policy uses spare bandwidth capacity (due to previously downloaded fragments) to guard against future quality switches.

C. Client-proxy cooperation

Perhaps one of the main problems with the basic cache policies and content-aware prefetching policies is that the quality selection choices made by the clients are independent of the content on the proxy. For example, in a scenario in which a client is quickly obtaining fragments cached by the proxy, the client may be tempted to increase the quality it is requesting, even in cases where the proxy does not have the higher encoding and there may not be sufficient bandwidth between the proxy and the server to quickly obtain this content.

In this section we present a cooperative policy class that allows fragment selection and prefetching decisions to be made based on information shared between the proxy and the client. By cooperation, the goal is to improve both the hit rates at the proxy cache and the viewer quality at the client.

Consider a system in which the client continually shares its buffer occupancy with the proxy and the proxy shares the map of fragments that it has access to with the client (for the particular file that the client currently is downloading). This information can (i) be shared at the beginning of the download and then updated as the cache content changes, or (ii) the proxy can continually update the client about the cache content for all fragments belonging to some upcoming time window from the current download point. In either case the client can give preference to downloading fragments that the proxy has. Among the fragment qualities stored on the proxy, we assume that the client would select the fragment quality with the most similar quality to that suggested by its rate-adaptation scheme (giving strict preference to lower qualities, if the intended quality is not available).[2] Now, given the clients request we consider two policy versions for the prefetching actions taken by the proxy.

- **Buffer oblivious:** This policy does not consider the client buffer state. Instead, it blindly uses the *priority-based* prefetching policy from the previous section.
- **Buffer aware:** This policy consider the client's current buffer occupancy. If the buffer occupancy is below a threshold $T \leq T_{min}^{buf}$ we use the following prioritization order: (i) current quality, (ii) one quality level below, (iii) two quality levels below, and (iv) no prefetching. On the other hand, if the client has higher buffer occupancy, we use the standard *priority-based* prefetching policy.

For the purpose of our performance evaluation, we developed a proof of concept implementation of the above policies. For this we had two options. We could either modify the player's source code, or build a wrapper around the player which intercepts and modifies the client requests. Due to a number of constraints with the flexibility of the SMP software, we selected the second approach, and make use of Tcpcatcher[3],

[2]Policies that request the fragments stored at the proxy with a probability p, and otherwise (with a probability $(1-p)$) pick whatever fragment is selected by its rate-adaptation mechanism are of course also possible.

[3]TcpCatcher. http://www.tcpcatcher.fr/, Sept. 2012

a proxy based monitoring software which provides handles to sniff and change packet contents on the fly.

At a high-level, the client connects to the Tcpcatcher port, and this port connects to the child port of our modified Squid proxy. This allows us to modify the client requests on the fly, such as to take into account the proxy contents. The client software is modified to send the buffer occupancy once a second, and the proxy sends the required fragment information to our Tcpcatcher module, which makes the decisions on behalf of the client. While our solution is non-optimal, in that SMP 2.0 is oblivious to any alterations to its requests, and a production implementation that modifies the player code could do additional enhancements to the rate adaption algorithm so as to achieve finer grained control, we do not expect any major performance differences with such implementations.

D. Performance evaluation

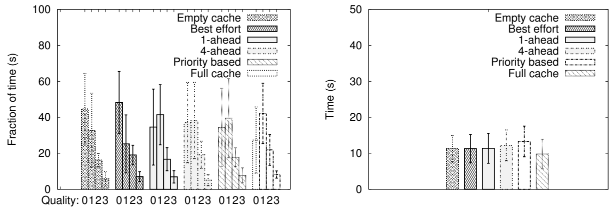
We now present the results of the experiments with different proxy cache configurations and policies. Similar to for the non-proxy case, we have found that our conclusions do not appear to depend on the loss rate and round-trip times. Instead, the major factor is the bandwidth variations and which link is the bottleneck. Due to these reasons and to save space, in the following, we present results only for the case in which the proxy-server RTT is 50ms, the client-proxy RTT is 6ms, and client uses the default buffer configuration $T_{min}^{buf}/T_{max}^{buf} = 4/6$.

1) Client-proxy bottleneck: Consider first the case when the bottleneck is between the client and proxy. For this case, we show results for when the proxy-server connection is not constrained, but the available client-proxy bandwidth follows the synthetic scenario with fast bandwidth variations.

To allow a fair comparison between policies, we replay the synthetic traces such that each of the policies sees exactly the same overall bandwidth conditions and clients arrive at the same time instances in each experiment. At the same time we make sure that each of the ten clients accessing the video, see different bandwidth conditions at each time instance of their individual download sessions. This way, the n^{th} client in each experiment sees exactly the same bandwidth conditions, but different than the m^{th} client, where $m \neq n$.

Figure 7 shows the averaged viewer quality and stall times over an experiment with ten clients for each of the six policies defined in Sections V-A and V-B. Note that despite improvements, the performance differences between the full cache policy (best case) and empty cache policy (worst case) are relatively small.[4] This suggests that proxies, regardless of policy, can provide only very limited performance advantages in the case the client-proxy link is the bottleneck.

Having said this, it should be noted that the simple 1-ahead prefetching policy is able to achieve most of these advantages. In contrast, the basic best effort policy performs very similar to



(a) Quality level (b) Stall time

Fig. 7. Comparison between baseline and content-aware proxy policies, and the bottleneck is between the clients and proxy.

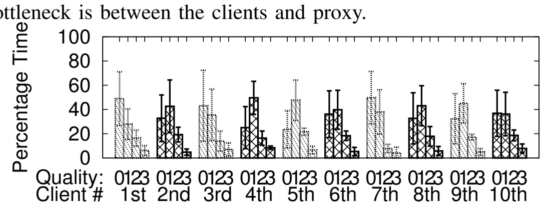
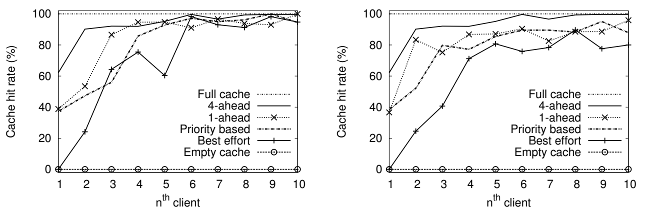


Fig. 8. Observed quality levels over n subsequent client downloads when using 1-ahead prefetching with client-proxy bottleneck.



(a) Client-proxy (b) Proxy-server

Fig. 9. Hit rate as a function of the number of previous downloads, when either client-proxy or proxy-server bottleneck.

the empty cache case. Our results also suggest that while there are performance improvements to prefetching, there is little gain to excessive prefetching. In fact, some clients experienced performance drops due to excessive prefetching clogging up the TCP pipeline.

To illustrate the longitudinal quality benefits of cache assistance for this scenario, Figure 8 shows the observed quality as a function of the number of previous clients that has downloaded the content when using the 1-ahead prefetching policy. The results here are the average statistics over five runs, each with ten clients. While the high variability in quality that different clients observe makes it more difficult to compare and distinguish the absolute quality levels observed by each client, it is clear that there is a longitudinal quality improvement benefiting clients arriving later.

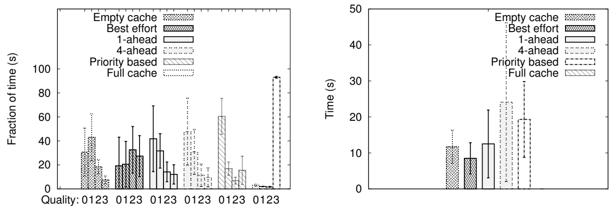
We next consider the cache hit rate of the different policies. Figure 9(a) shows the cache hit rate as a function of the number of previous clients that have downloaded the video. Note that after two-three clients most policies have a hit rate of 80-90%. For the best effort policy it takes roughly five clients before the hit rate surpass 90%.

When interpreting the somewhat higher overall hit rates observed by the prefetching policies, it should be noted that there may be some bandwidth wasted when fragments are prefetched. The first six rows of the left-hand column of Table I summarize the bandwidth usage (measured as the total transferred bytes) between the proxy and the server for these experiments. We see that both the best effort (59.5 Mb) and 1-ahead (64.9 Mb) achieve similar bandwidth usage savings

[4]To make sure that the small performance improvements were not due to optimizations made at the server we took a closer look at the service times of individual fragments, when served by the server or the (full) proxy cache respectively. In fact, for the case with the same RTT to both the server and the proxy, we observed a smaller time to first byte ($\bar{x} = 8.30ms; \sigma = 15.03ms$) for the full cache than for the server ($\bar{x} = 16.10ms; \sigma = 7.85ms$). For the time to last byte, the differences were negligible: full cache ($\bar{x} = 2.71s; \sigma = 1.92s$) and server ($\bar{x} = 2.81s; \sigma = 1.97s$).

TABLE I. BANDWIDTH USAGE BETWEEN PROXY AND SERVER FOR THE POLICIES DEFINED IN SECTIONS V-A AND V-B.

	Bandwidth bottleneck location	
Policy	Client-proxy	Proxy-server
Empty cache	169.2 Mb	184.9 Mb
Best effort	59.5 Mb	94.7 Mb
1-ahead	64.9 Mb	65.9 Mb
4-ahead	77.09 Mb	77.85 Mb
Priority	81.3 Mb	72.2 Mb
Full cache	–	–
Cooperation/oblivious	89.97 Mb	97.2 Mb
Cooperation/aware	103.7 Mb	105.77 Mb



(a) Quality level (b) Stall time

Fig. 10. Comparison between baseline and content-aware policies; proxy-server bottleneck.

compared to the empty cache policy (169.2 Mb), while in both cases delivering the clients higher quality encoding. This suggests that the extra fragments obtained through prefetching often may be useful for later requests (as seen by the higher hit rates).

2) Proxy-server bottleneck: We next consider the case when the bottleneck is between the proxy and the server. We show results for when the client-proxy connection is not constrained, but the available bandwidth of the proxy-server link follows that of the synthetic scenario with fast bandwidth variations.

Figure 10 shows the averaged viewer quality and stall times for each of the six policies defined in Sections V-A and V-B. Comparing the full cache policy and empty cache policy, we note that there is very large performance potential for proxy caching in this scenario. Overall, the full cache policy has significantly superior performance compared to all the other policies. It has no stall times and is able to serve the clients almost entirely at the highest quality.

It is encouraging that a significant performance improvement is achieved with the best effort policy alone. This policy is easy to implement and does not require content awareness. Yet, it is able to reduce the stall times and allow the video to be played at a much higher quality than with the empty cache policy (or no cache, for that matter).

It may, however, be discouraging that none of the three basic prefetching policies are able to improve on this. In fact, rather the opposite, these policies all achieve worse performance than the best effort policy. The problem with these policies is that the prefetching itself clogs up the bottleneck between the proxy and server. This illustrate that more careful policies, such as our cooperative policy, are required to take better advantage of the cache and its bandwidth.

The limited success of the prefetching policies can be further observed when considering the cache hit rates. Figure 9(b) shows the hit rate as a function of the number of previous clients. Comparing with Figure 9(a), we note that the hit rate in this scenario is significantly lower than the hit rate seen when the bottleneck is between the client and proxy.

TABLE II. AVERAGE PLAYBACK QUALITY (MEASURED IN KBPS) OF EXAMPLE POLICIES UNDER DIFFERENT SCENARIOS.

		Policy					
	Scenario	Empty	Best effort	1-ahead	4-ahead	Priority	Full
Client-proxy	Bus	649	766	742	771	773	790
	Ferry	618	822	808	882	800	833
	Metro	344	428	429	395	433	441
	Tram	414	428	420	453	415	465
	Loss (2.5%)	474	509	553	521	480	508
	Loss (7.5%)	308	306	296	311	311	324
	Default	494	508	533	521	539	577
Proxy-server	Bus	852	1050	794	673	1172	1249
	Ferry	803	997	811	670	1174	1249
	Metro	446	685	393	389	663	1249
	Tram	464	680	436	416	692	1249
	Loss (2.5%)	502	730	510	495	413	1253
	Loss (7.5%)	305	597	403	407	341	1253
	Default	550	790	541	505	503	1249

Due to space constraints we can only include a limited number of representative results for our default scenario. Table II summarizes the average playback quality for the corresponding experiments for the real world scenarios and experiments with larger loss rates and RTTs. In general, we have found that our conclusions hold true also for these scenarios.

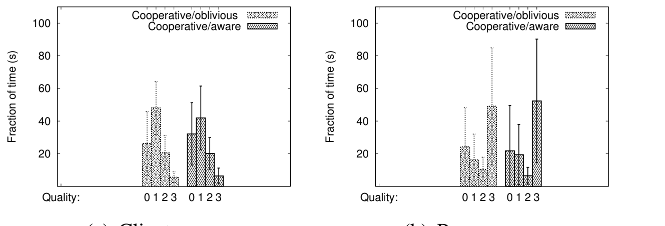
3) Client-proxy cooperation: Motivated by the large performance gap between the full cache and best effort policy, we consider the performance under the client-proxy cooperative policies. Ideally, these policies should help improve performance by adjusting the requested quality level based on the content in the cache.

Figures 11(a) and 11(b) show the quality levels for these policies for the cases when the client-proxy link is the bottleneck and when the bottleneck is between the proxy and the server, respectively. The stall times are similar to those of the 1-ahead policy. Comparing these results with those in Figures 7 and 10, note that the cooperative policies can significantly improve the viewed video quality. Not surprisingly, the largest improvements are achieved when the bottleneck is between the proxy and server. In this case the policies are able to effectively leverage the spare proxy-server bandwidth enabled by the boosted hit rates, to prefetch fragments that are likely to help the current or future clients.

The high playback quality achieved by the buffer oblivious cooperative policy is particularly impressive for the case the bottleneck is located between the proxy and server. For this case, as shown in Table I, the buffer oblivious policy uses almost the same bandwidth (97.2 Mb) between the proxy and server as the best effort policy (94.7 Mb), but delivers much higher quality to the client.

To illustrate the performance improvements observed by later clients, Figure 12 shows the observed quality as a function of the number of previous clients. While this data is noisy due to the difference in bandwidth observed by each client, it is clear that the cooperative policy is able to load the cache with content that future clients can benefit from, without hurting the performance of the early clients.

4) Buffer size: For our final experiments, we revisit the question about how the player buffer size may impact our conclusions regarding which policies are the most advantageous



(a) Client-proxy (b) Proxy-server

Fig. 11. Client-proxy cooperation experiments, when client-proxy and proxy-server bottleneck.

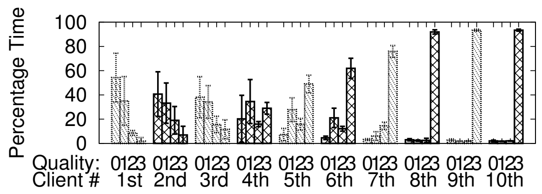


Fig. 12. Observed quality levels over n subsequent client downloads when the copperative buffer oblivious policy is used, and the bottleneck is between the proxy and server.

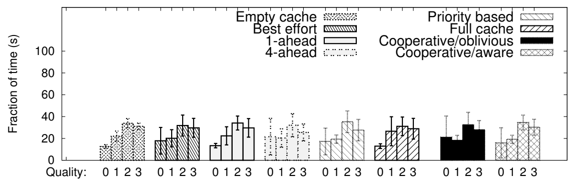
in each scenario. In general, and as shown in Section IV, we have found that a larger buffer size reduces the amount of buffer interruptions, increases the average video quality, and helps improve the overall video playback experience.

Figures 13 and 14 summarize the video quality and stall times, respectively, when $T^{buf}_{min}/T^{buf}_{max} = 12/20$ (rather than the default buffer size of 4/6) for the cases when the bandwidth bottleneck is between the client-server and proxy-server. In addition to overall better performance, it is interesting to note that the simpler best effort policy benefits the most from the larger buffers. The best effort policy is able to achieve very small stall times and relatively high video quality. In fact, in many cases it outperforms the more complex prefetching policies. These results suggest that a larger client buffer reduces the importance of advanced proxy policies. With larger client buffers resulting in unnecessary data transfers when users terminate their viewing early, advanced policies may therefore be most attractive in wireless environments, with relatively expensive links with battery-powered mobile devices.

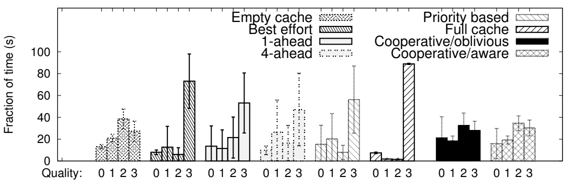
The much shorter overall stall times for the best effort policy are due to two compounding factors. First, the best effort policy sees a much bigger reduction in the number of stalls per minute (by a factor of 8.4 compared to 2.6 for the 1-ahead policy, for example) and in the average stall time per playback interruption (by a factor 2.0 while the 1-ahead policy increases by 1.3). These two factors have a multiplicative effect, resulting in much smaller overall stall times.

VI. Related works

Motivated by client heterogeneity and variations in the available bandwidth, video streaming solutions have often been required to offer multiple encodings. For example, layered encodings have been used with adaptive adjustment of the number of layers each client is served (using UDP) so as to be TCP-friendly [7], [22]. Such quality adaptation schemes have also been extended to incorporate cache replacement and prefetching [23], including techniques that use segment-based prefetching [9] or fill gaps in the cached video content [27].

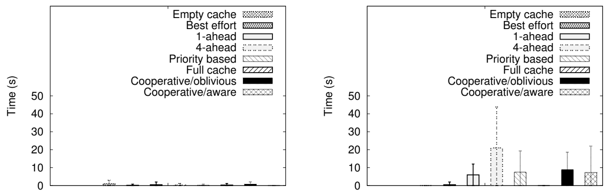


(a) Client-proxy



(b) Proxy-server

Fig. 13. Summary of quality level statsitics under different policies when using larger client buffer.



(a) Client-proxy (b) Proxy-server

Fig. 14. Summary of stall times under different policies when using larger client buffer.

Segment-based proxy policies have also been considered for non-adaptive video streaming [8], [16]. None of these works consider HAS.

Although HAS is widely used [1], [10] there are few performance studies. Akhshabi et al. [3] compare the performance of three players: MSS, Netflix, and an older version of the SMP media player used here. As the first two players are proprietary, they are limited to using traffic measurements to infer the performance at the players. Riiser et al. [24] performed a similar study in which they used real-world traces to model the bottleneck link conditions and then evaluated performance through observation of the packets delivered to the client network interface.

While the results by Akhshabi et al. [3] clearly show that the performance of the two commercial players is much better than that of SMP, our results suggest that the much larger buffer size may have played a big role in their conclusions. For example, MSS uses a buffer size of 20-30 seconds [3], [6], while Netflix can buffer much more than that [3]. We have also found that SMP 2.0 significantly improves on the previous versions of the SMP player.

Perhaps the work by Benno et al. [6] is most closely related to ours. Benno et al. [6] present results on how proxy caches interfere with the MSS rate detection algorithm, as well as how cross-traffic can influence the player performance. Similar to the above studies they were limited to using trace-based estimates of the player performance, and only considered very basic proxy scenarios.

Other works have taken a closer look at the rate adaptation algorithms [18], quality selection policy when using scalable video codes [4], the performance at the servers under HAS workloads [26], the traffic patterns they generate [21], or the impact of competing players and/or TCP flows [2], [11]. Finally, Guota et al. [14] characterize the caching opportunities of HAS content in a national mobile network.

VII. Conclusion

With HAS responsible for large traffic volumes, network and service providers may consider integrating customized HAS-aware proxy policies. In this paper we consider the potential performance impact of such optimizations.

We present an experimental framework using open source Squid proxies and the most recent Open Source Media Framework (OSMF). The open source software allows us to implement and test content-aware policies, while measuring the performance as seen at the client.

Using this framework, we first present a scenario-based performance evaluation of the latest version of the OSMF player, and show that this player has substantially improved performance in comparison to the versions employed in previous work. We then present a thorough longitudinal evaluation in which we evaluate a range of different policy classes that may be employed in proxy-assisted systems and provide insights to the effectiveness of different policies and their performance tradeoffs. We quantify the benefits using different proxy-assisted solutions, including basic best effort policies and more advanced content quality aware pre-fetching policies. Finally, we present and evaluate a cooperative framework in which clients and proxies share information to improve performance. Our results show that the bottleneck location and network conditions play central roles in which policy choices are most advantageous, and that careful proxy design and policy selection is important when trying to enhance HAS performance in edge networks.

Future work includes the development and testing of adaptive proxy policies that make prefetching decisions based on estimates of the available bandwidth for both the client-proxy and proxy-server connections. Such policies could, for example, perform more aggressive prefetching during times when there is spare bandwidth between the proxy and server.

VIII. Acknowledgements

This work was supported by funding from Center for Industrial Information Technology (CENIIT) and the Swedish National Graduate School in Computer Science (CUGS) at Linköping University, the Natural Sciences and Engineering Research Council (NSERC) of Canada, and National ICT Australia (NICTA).

References

[1] Sandvine global Internet phenomena Report - 1h2012. Technical report, Sandvine Incorporated ULC, April 2012.

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

[3] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proc. ACM MMSys*, San Jose, CA, Feb. 2011.

[4] T. Andelin, V. Chetty, D. Harbaugh, S. Warnick, and D. Zappala. Quality selection for dynamic adaptive streaming over HTTP with scalable video coding. In *Proc. ACM MMSys*, Chapel Hill, NC, Feb. 2012.

[5] A. C. Begen, T. Akgul, and M. Baugher. Watching video over the Web: Part 1: Streaming protocols. *IEEE Internet Computing*, (15):54–63, 2011.

[6] S. Benno, J. O. Esteban, and I. Rimac. Adaptive streaming: The network HAS to help. *Bell Lab. Tech. J.*, 16(2):101–114, Sept. 2011.

[7] J. Byers, G. Horn, M. Luby, M. Mitzenmacher, and W. Shaver. Flid-dl: congestion control for layered multicast. *IEEE Journal on Selected Areas in Communications*, 20:1558–1570, Oct. 2002.

[8] S. Chen, B. Shen, S. Wee, and X. Zhang. Segment-based streaming media proxy: Modeling and optimization. *IEEE Transactions on Multimedia*, 8(2), Apr. 2006.

[9] S. Chen, H. Wang, B. Shen, S. Wee, and X. Zhang. Segment-based proxy caching for Internet streaming media delivery. *IEEE Multimedia*, 12(3):59–67, 2005.

[10] J. Erman, A. Gerber, S. Sen, O. Spatscheck, and K. Ramakrishnan. Over the top video: The gorilla in cellular networks. In *Proc. ACM IMC*, Germany, Nov. 2011.

[11] J. Esteban, S. Benno, A. Beck, Y. Guo, V. Hilt, and I. Rimac. Interactions between HTTP adaptive streaming and TCP. In *Proc. NOSSDAV*, Toronto, ON, Canada, June 2012.

[12] J. L. Feuvre, C. Concolato, and J. C. Moissinac. Gpac, open source multimedia framework. In *Proc. ACM MM*, Germany, Sept. 2007.

[13] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. Youtube traffic characterization: A view from the edge. In *Proc. ACM IMC*, San Diego, CA, Oct. 2007.

[14] A. Gouta, D. Hong, A.-M. Kermarrec, and Y. Lelouedec. HTTP adaptive streaming in mobile networks: Characteristics and caching opportunities. In *Proc. IEEE MASCOTS*, San Francisco, CA, Aug. 2013.

[15] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proc. ACM CoNEXT*, Rome, Italy, Dec. 2009.

[16] S. Khemmarat, R. Zhou, D. Krishnappa, and M. Z. L. Gao. Watching user generated videos with prefetching. In *Proc. ACM MMSys*, San Jose, CA, Feb. 2011.

[17] M. Levkov. Video encoding and transcoding recommendations for HTTP dynamic streaming on the Adobe® Flash® platform. Technical report, Oct. 2010.

[18] C. Liu, I. Bouazizi, and M. Gabbouj. Rate adaptation for adaptive HTTP streaming. In *Proc. ACM MMSys*, San Jose, CA, Feb. 2011.

[19] A. Mahanti, N. Carlsson, A. Mahanti, M. Arlitt, and C. Williamson. A tale of the tails: Power-laws in Internet measurements. *IEEE Network*, 27(1):59–64, Jan/Feb. 2013.

[20] C. Müller and C. Timmerer. A vlc media player plugin enabling dynamic adaptive streaming over HTTP. In *Proc. ACM MM*, Scottsdale, AZ, 2011.

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

[22] R. Rejaie, M. Handley, and D. Estrin. Layered quality adaptation for Internet video streaming. *IEEE Journal on Selected Areas in Communications*, (18):2530–2543, Dec. 2000.

[23] R. Rejaie and J. Kangasharju. Mocha: A quality adaptive multimedia proxy cache for Internet streaming. In *Proc. NOSSDAV*, Port Jefferson, NY, June 2001.

[24] 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 (MoVid)*, Chapel Hill, NC, Feb. 2012.

[25] L. Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27:31–41, 1997.

[26] J. Summers, T. Brecht, D. L. Eager, and B. Wong. To chunk or not to chunk: Implications for HTTP streaming video server performance. In *Proc. NOSSDAV*, Toronto, ON, Canada, June 2012.

[27] M. Zink, J. Schmitt, and R. Steinmetz. Layer-encoded video in scalable adaptive streaming. *IEEE Trans. on Multimedia*, 7(1):75–84, Feb. 2005.