# Quality-adaptive Prefetching for Interactive Branched Video using HTTP-based Adaptive Streaming[*]

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

Interactive branched video that allows users to select their own paths through the video, provides creative content designers with great personalization opportunities; however, such video also introduces significant new challenges for the system developer. For example, without careful prefetching and buffer management, the use of multiple alternative playback paths can easily result in playback interruptions. In this paper, we present a full implementation of an interactive branched video player using HTTP-based Adaptive Streaming (HAS) that provides seamless playback even when the users defer their branch path choices to the last possible moment. Our design includes optimized prefetching policies that we derive under a simple optimization framework, effective buffer management of prefetched data, and the use of parallel TCP connections to achieve efficient buffer workahead. Through performance evaluation under a wide range of scenarios, we show that our optimized policies can effectively prefetch data of carefully selected qualities along multiple alternative paths such as to ensure seamless playback, offering users a pleasant viewing experience without playback interruptions.

## 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); Branched video; Multipath/nonlinear streaming; Seamless playback

## 1. INTRODUCTION

On-demand video streaming has gained tremendous popularity and currently accounts for a large fraction of Internet traffic [24]. In contrast to the original UDP-based services, most of today's on-demand video streaming services are using HTTP over TCP. The use of HTTP-based streaming simplifies caching and facilitates simple traversal of NATs and firewalls. With the introduction of HTTP-based adaptive streaming (HAS) protocols, the player can now also easily adapt the streaming quality based on the user's current bandwidth conditions [3, 5].

Another important development has been the increased level of personalization on the Web. As companies and users become increasingly used to personalized services, we expect increasing personalization of video. For example, a viewer may find a movie too sad, too violent, or too scary. What if the content provider could customize the video playback sequence based on the taste of each user?

Interactive branched video (previously also called "nonlinear" and "multipath" video) that allow users to traverse different plot sequences, depending on their interactions with the video (e.g., by pushing a button on a keyboard, TV remote, or with the help of a mouse click) have been proposed in the research literature [11, 12, 17, 27], and explored through clever (but non-trivial) use of navigation captions in YouTube videos. Recently, Interlude[1] even launched a Web-based service that offers content creators an easy way to create interactive videos hosted by the company.

While these types of media provide many advantages and opportunities, they also present new challenges. For example, with the user playback experience (at least of linear video) being dominated by the number and durations of any potential playback interruptions [9], branched video presents a unique challenge as the player may not know in advance which of the potential paths a user will take. Therefore, video must be prefetched along multiple paths to ensure that playback is seamless, without playback interruptions, even when users defer their choices to the last possible moment.

We argue that HTTP-based Adaptive Streaming (HAS) provides an excellent framework for implementing interactive branched video streaming (Section 2). Using HAS together with a simple media description and path tracking mechanism (i) provides the creator with full flexibility when designing personalized content, (ii) does not add any additional overhead on the content servers storing the media files compared to delivery of regular linear non-interactive media, and (iii) ensures that the solution is TCP friendly, easily traverses firewalls/NATs, and can leverage all of the benefits of content replication and proxy caches.

In this paper, we describe a design and full implementation of such a HAS-based interactive branched video stream-

[1]Interlude, `http://www.interlude.fm`, Mar. 2014.

*MM'14,* November 03 - 07, 2014, Orlando, FL, USA
Copyright 2014 ACM 978-1-4503-3063-3/14/11 ...$15.00.
http://dx.doi.org/10.1145/2647868.2654951.

ing system. Using a relatively simple model and this implementation, we explore the key question of what policy a branched video player should use when deciding which video chunks to download when. Such a policy is analogous to the rate adaptation policy in HAS, but is made greatly more complex by uncertainty regarding future user path selections. The contributions of this paper are as follows.

We first develop a simple analytic model that allows definition of the basic prefetch problem as an optimization problem in which the expected playback quality is maximized, while avoiding playback interruptions. Within this framework we argue that it is optimal to download back-to-back video chunks in a round-robin manner, and that there exists a natural tradeoff between playback quality and the use of additional parallel TCP connections to build up buffer workahead and ensure seamless playback.

Second, based on these findings, we design optimized policies that determine (i) when different chunks of the video should be downloaded, (ii) at what quality level each such chunk should be encoded with, and (iii) how to manage playback buffers and TCP connections such as to ensure a smooth playback experience without excessive workahead. By extending the existing buffer management policies used for HAS our policies carefully balance the buffer workahead needed to ensure seamless playback against potential wasted bandwidth from prefetching data along non-used paths or for an early terminated session.

Finally, we present the design, implementation (Section 4) and experimental evaluation (Section 5) of our framework. In addition to providing concrete evidence that the optimized policies perform well under a wide range of scenarios, our experiments provide insights into the importance of careful adaptive policies. For example, it is shown that the use of parallel connections can be particularly valuable when building workahead in environments with much competing traffic, and our optimized prefetching policies together with a capped workahead policy is shown to provide a good tradeoff between ensuring a smooth playback experience and avoiding excessive workahead.

The remainder of the paper is organized as follows. Section 2 presents our general HAS-based branched video framework. Section 3 defines our system model, problem formulation, and optimized prefetching policies. Section 4 describes our system design implementation. Our experimental evaluation is presented in Section 5. Section 6 discusses related work, and Section 7 concludes the paper.

## 2. PERSONALIZED CONTENT DELIVERY

### 2.1 Chunks, Segments, and Branch Points

HAS players typically either use byte-range requests or split the video file into *chunks*, each addressable with a unique URL. While the general design in this paper is applicable to both types, we will present a chunk-based solution. A manifest file providing URLs for individual chunks is used to bootstrap the player. Each chunk is typically 2-5 seconds long, and chunk boundaries coincide for the different encodings, allowing the client to switch to any available encoding at chunk boundaries.

Branched video, as we define it here, generalizes traditional linear HAS video in that it allows (i) the video designer to define arbitrary playback sequences through the underlying linear video, and (ii) the users to select among multiple alternative playback sequences. In order to allow for such customization and interactivity, the designer can define both linear and nonlinear *segments*. These are linked together using *branch points* at which the user can chose among multiple path options [17].

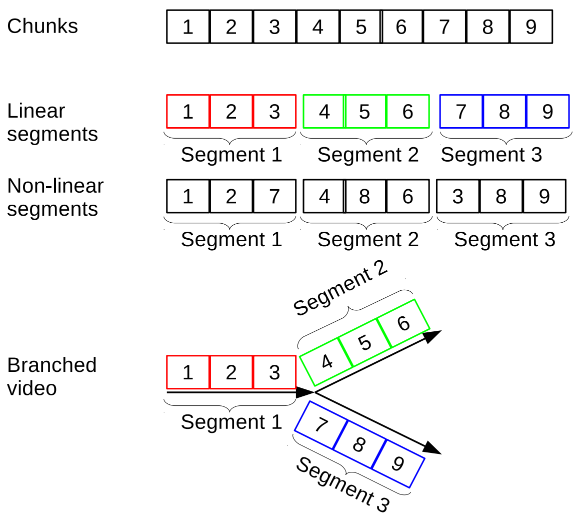


**Figure 1: Illustration of terminology**

Figure 1 illustrates our terminology. Here, as with regular HAS video, a chunk contains a small fixed duration of the video and is defined within the video manifest file. Each segment (linear or nonlinear) consists of a sequence of chunks that are played in the specified order without the need for any user interaction. Branch points are used to give users the option of alternative video playback paths. While playing the segment leading up to an applicable branch point, the user is given the option of which of several possible next segments the player should play next.

### 2.2 Branched Video Structure

Our client-driven design does not add any client tracking or additional processing overhead to the HAS servers. The designer simply specifies segments and branch point rules in a potentially-personalizable, text-based metafile, separate from the manifest file that comes with the underlying HAS media. The player is responsible for determining which branch points to consider and chunks to download.

Segments can be specified as any arbitrary sequence of chunks and the applicable branch points can be defined using rich branch point rules that specify (i) the playback path that the client must have taken for the branch point rule to be applied, (ii) the options for what path to follow after the branch point, (iii) the weight associated with each path choice, and (iv) the message to be displayed to the user whenever that branch point is in play. When playing the video, our player keeps track of the playback sequence of the user and checks the applicability of each of the branch point rules associated with the next upcoming branch point. Among all matching branch point rules, the player picks the one with the longest match. In the remainder of this paper we will focus on how the player performs prefetching, given the outcome from applying this rule.

Consider a branched video that consists of $E = |\mathcal{E}|$ segments and $B = |\mathcal{B}|$ branch points. Here, each segment $e \in \mathcal{E}$ is of playback length $L_e$ and consists of $n_e$ chunks each with playback duration $l_{e,i}$, where $1 \leq i \leq n_e$. Each chunk can

Table 1: Notation for branched video

| Symbol | Definition |
|---|---|
| $\mathcal{E}$ | Set of all segments $e$ (linear or nonlinear) |
| $L_e$ | Media playback duration in segment $e$ |
| $n_e$ | Total number of chunks in segment $e$ |
| $l_{e,i}$ | Total media playback length of chunk $i$ in segment $e$ |
| $\mathcal{Q}$ | Set of quality encodings |
| $q_{e,i}$ | The quality encoding of chunk $i$ in segment $e$ |
| $\mathcal{B}$ | Set of all branch points $b$ |
| $\mathcal{E}^b$ | Set of branch edges $e$ associated with branch point $b$ |
| $w_e^b$ | Relative weight for branch-point edge $e$ of branch point $b$ |
| $\Delta^u$ | Protocol threshold time used by client $u$ |
| $t_i^s$ | Start of download of chunk $i$ |
| $t_i^c$ | Download completion of chunk $i$ |
| $t_i^d$ | Playback deadline of chunk $i$ |
| $l_i$ | Total media playback duration of the $i^{\text{th}}$ chunk among those in segment $e$ and those immediately following branch point $b$ |
| $q_i$ | The quality encoding of the $i^{\text{th}}$ chunk among those in segment $e$ and those immediately following branch point $b$ |
| $r_i$ | Estimated per-connection download rate between download initiation times $t_i^s$ and $t_{i+1}^s$ |
| $c_i$ | Number of parallel connections between download initiation times $t_{i-1}^s$ and $t_i^s$ |
| $R_i(c)$ | Estimated total download rate between the initiations of the $i^{\text{th}}$ and $(i+1)^{\text{st}}$ chunk downloads, when using $c$ connections |

be downloaded in one of $Q = |\mathcal{Q}|$ qualities, where $\mathcal{Q}$ is the set of qualities, each corresponding to a particular video bitrate. We use $q_{e,i}$ to denote the quality (bitrate) of chunk $i$ of segment $e$, and hence the size of this chunk is $q_{e,i}l_{e,i}$.

We denote by $\mathcal{E}^b$ the set of branch options (in particular, possible choices for the next segment to play) available to a particular client reaching branch point $b \in \mathcal{B}$. Each is given a weight $w_e^b$. In the following, we will assume that all weights are normalized, such that $\sum_{e \in \mathcal{E}^b} w_e^b = 1$, and the weights reflect relative priorities (e.g., the probability that a path is selected). Table 1 summarizes our notation.

## 3. PLAYER MANAGEMENT

### 3.1 Problem Description

We design optimized prefetching policies and buffer management schemes that (i) provide uninterrupted playback with seamless switches from one segment to another, and (ii) given such uninterrupted playback, maximize the playback quality. Our HAS-based solution takes advantage of parallel TCP connections to build up a workahead buffer to keep unnecessary stalls at a minimum, but must also make careful tradeoffs among workahead, playback quality, and the potentially wasted bandwidth associated with downloading chunks the client later does not use.

As an illustration of these design choices, consider first Figure 1. Assume that the player is currently playing segment 1. To ensure seamless playback when reaching the branch point, the first chunk of each path choice (i.e., chunks 4 and 7) must have been retrieved before reaching the branch point. To make good prefetch decisions, the client must keep track of how much time there is until the branch point, and how much bandwidth is available, so as to determine when and at what quality each chunk should be downloaded.

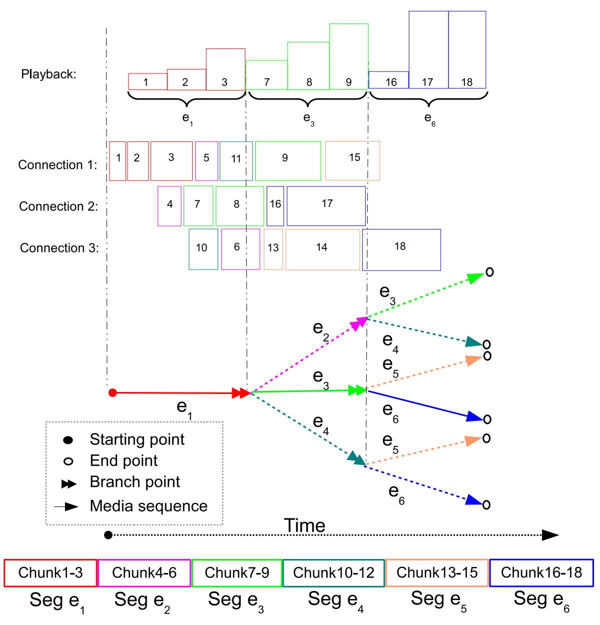


**Figure 2: Round-robin parallel downloading**

Figure 2 illustrates a more complex branched video structure in which the client must make multiple path decisions (bottom half of the figure), the download schedule (middle) and the playback schedule (top) for an example player implementing a basic policy. Note how all chunks along the selected (solid lines) playback path are obtained in time of their playback deadlines (i.e., the time by which the player needs that chunk for it to be played without a stall), how up to three parallel connections are used to build up workahead and prefetch the first chunk of each potential path choice (chunks 4, 7, and 10 for the first branch point, and then chunks 13 and 16 for the second branch point) before reaching the branch points, as well as how different qualities are selected for the different chunks depending on how much workahead is available before each (potential) playback deadline. In this scenario a simple round-robin schedule is used for determining which chunk to download next.

This example illustrates the importance of careful prefetching. More generally, important design considerations in determining a good download schedule include:

- **Download ordering and playback quality.** With different chunks being associated with paths of different likelihood and with different delay constraints, both the order of downloads and the chosen quality for each chunk play an important role in ensuring seamless playback at the highest possible playback quality.
- **Concurrent downloads.** To allow improved download speeds and prefetching we allow multiple parallel connections. However, when doing so, we must consider the tradeoff between improving the overall download rate and meeting individual playback deadlines.
- **Capped workahead.** To avoid excessive workahead and wasted bandwidth usage when a user terminates a session early, most HAS players use on-off control and do not allow the buffer to fill up beyond a certain

threshold. In the context of branched video this concept must be extended to take into account the number of path options at each encountered branch point. Depending on the underlying structure, our player sets the target buffer size dynamically.

The player must determine when chunks should be downloaded, at what qualities, and how much parallelism should be utilized, such as to maximize the playback quality and meet playback deadlines. Given a network constraint on how much data can be downloaded over a given time period, there is a tradeoff between quality (higher quality implies larger chunks) and the number of chunks that can be downloaded. The stall probability depends on network conditions and the prefetching policy (such as concerning prefetching of chunks from after a branch point).

## 3.2 Objectives and Constraints

In this section we formulate the prefetching problem as an optimization problem. Without loss of generality, we consider a client downloading a single segment $e$ that has been determined to have an upcoming branch point $b$. We want to maximize the playback quality of the chunks of the current segment $e$ and the first chunk of each of the $|\mathcal{E}^b|$ different path choices, conditioned on each chunk, including the first chunk of each potential segment following the branch point, being downloaded prior to its (potential) playback deadline.

**Objective function:** Assuming that a client picks path $e' \in \mathcal{E}^b$ at branch point $b$ with a probability $w^b_{e'}$, we can formulate our objective function as:

$$\text{maximize} \sum_{i=1}^{n_e} q_{e,i} l_{e,i} + \sum_{e' \in \mathcal{E}^b} w^b_{e'} q_{e',1} l_{e',1}. \tag{1}$$

**Round-robin ordering:** Without loss of generality, we enumerate the chunks associated with a segment $e$ and the first chunk in all potential following segments as follows: (i) the chunks of segment $e$ are enumerated from 1 to $n_e$, and (ii) the first chunk from all of the segments following the branch point $b$ are enumerated from $n_e+1$ to $n_e+|\mathcal{E}^b|$ with chunk $n_e+1$ having the highest weight $w^b_{e'}$ and $n_e+|\mathcal{E}^b|$ the lowest. With this numbering, let $q_i$ and $l_i$ be the quality and chunk length, respectively, of the $i^{\text{th}}$ chunk of the combined segment-branch-point pair $\{e, b\}$.

Clearly, the downloads of the chunks in segment $e$ should be initiated before those for chunks following the branch point. Furthermore, to avoid stalls, in the case where all of the probabilities $w^b_{e'}$ are significant, it is optimal to use a round-robin ordering for workahead from after the branch point; i.e., the downloads for the first chunk from all of the possible following segments should be initiated, prior to initiating downloads of any subsequent chunks. These "first chunk" downloads should be initiated in order of the respective weights $w^b_{e'}$. Therefore, workahead downloads should be initiated in the same order as in our enumeration. Finally, for similar reasons, it may be desirable to choose non-increasing quality levels for successive workahead downloads.

Motivated by the above observations, in the following we only consider round-robin schedules following the above order. In fact, in the case that additional workahead is possible after downloading all the $|\mathcal{E}^b|$ chunks and our workahead rule (Section 3.4) allows for greater workahead beyond the branch point, we again use round-robin among the second chunk of the candidate segments.

With our chunk order and notation, we can rewrite the objective function as:

$$\text{maximize} \sum_{i=1}^{n_e} q_i l_i + \sum_{i=n_e+1}^{n_e+|\mathcal{E}^b|} q_i l_i. \tag{2}$$

**Playback deadlines:** Let $t^c_i$ and $t^d_i$ represent the time at which the client completes download of chunk $i$ and the playback deadline of chunk $i$, respectively.

Our optimization problem has the constraint that the download completion time $t^c_i$ of each chunk ($1 \le i \le n_e + |\mathcal{E}^b|$) must be before the playback deadline of that chunk:

$$t^c_i \le t^d_i = \begin{cases} \tau + \sum_{j=1}^{i-1} l_j, & \text{if } 1 \le i \le n_e \\ \tau + \sum_{j=1}^{n_e} l_j, & \text{if } n_e < i \le n_e + |\mathcal{E}^b| \end{cases} \tag{3}$$

Here, $\tau$ is used to denote the time at which the first chunk of segment $e$ begins playback. If time is measured relative to the time at which video download is initiated, then for the first segment on a client's playback path, $\tau$ corresponds to the startup delay. For subsequent segments, $\tau$ is given by the value of $\tau$ for the preceding segment on the client's playback path, plus the total playback duration of that segment.

**Startup delays:** For simplicity, we calculate the startup delay considering only the first segment $e$ on a client's playback path, and the subsequent branch point $b$. Further, we use the minimum possible startup delay when all downloaded chunks are downloaded sequentially at some minimum desired quality $q^*$. The startup delay $\tau$ can then be calculated as the maximum of $\max_{1 \le i \le n_e} [\sum_{j=1}^{i} \frac{q^* l_j}{R(1)} - \sum_{j=1}^{i-1} l_j]$ and $\sum_{j=1}^{n_e+|\mathcal{E}^b|} \frac{q^* l_j}{R(1)} - \sum_{j=1}^{n_e} l_j$, where $R(1)$ is the download rate when using a single TCP connection. Here, the first expression uses the tightest download completion time constraint over all of the chunks of segment $e$ (typically yielding a startup delay equal to the download time of the first chunk, $t^c_1$) and the second expression uses the constraint of obtaining the first chunk of each branch choice in time for playback. The same method is used again anytime that there is a playback interruption and $\tau$ must be re-calculated.

**Download times:** Constraint (3) is on download completion times. Here we relate these completion times to total download rate and the policy variables (number of parallel connections, prefetching policy decisions). We restrict attention to policies that only initiate new chunk downloads and open new TCP connections at the time of a download completion. Such policies simplify connection management.

With a measured total download rate of $R_i(c_i)$ between the initiations of the $i^{\text{th}}$ and $(i+1)^{\text{st}}$ chunk downloads, the average download rate per connection is equal to $r_i = R_i(c_i)/c_i$, where $c_i$ is the number of active parallel connections during this time period. Letting $t^s_i$ denote the time at which the download of chunk $i$ is initiated, and assuming that the per-connection rates $r_i$ are known, we can formulate the following conservation equation:

$$\sum_{j=i}^{i+k} r_j (t^s_{j+1} - t^s_j) = q_i l_i, \tag{4}$$

where we have assumed that $k$ new chunk downloads are initiated over the download time of the $i^{\text{th}}$ chunk; i.e., $t^c_i = t^s_{i+k}$. Here, the left-hand side gives the total amount of data downloaded on a single connection during the download time of chunk $i$, recognizing that the download rate on that connection will change with the number of active parallel connections. The right-hand side gives the chunk size.

**Download rates:** In practice, neither the rates $r_j$ nor any of the future download completion times are known at the time of a download completion, when the player must determine (i) how many parallel connections to use until the next completion, and (ii) at what quality any new chunk should be downloaded. At one extreme the client has no competition on the network bottleneck link. In this case $R_i(c_i)$ is independent of $c_i$ and $r_{i+1} = \frac{c_i}{c_{i+1}} r_i$. At the other extreme, the client has very many competing flows, $R_i(c_i)$ is approximately proportional to $c_i$, and $r_i$ can be assumed to be independent of $c_i$.

To help ensure that delay constraints are satisfied even under uncertainty, our policies use the conservative assumption that there will be no additional bandwidth gains from using additional TCP connections, but prioritize parallelism when possible without violating these delay constraints.

## 3.3 Prefetching Policies

The player initially opens one connection to the server, and subsequently opens additional connections at download completion time instants, depending on the policy. Idle connections are reused, and, as shown in Figure 2, HTTP GETs for different paths can share a single connection.

To determine the number of new chunks to request next, and the quality at which each of these chunks should be downloaded, we use our optimization problem formulation. At the completion of a chunk download, we pick the quality and number of parallel new downloads that maximizes the expected weighted playback quality, as defined by the objective function (2), and that satisfies all constraints.

We consider two policies that both restrict the number of candidate schedules to consider, but differ in their complexity and how aggressive they are. The first policy, called *Optimized non-increasing quality*, considers all possible candidate schedules that (i) do not exceed the maximum number of parallel connections allowed $C$, (ii) do not count on opening additional connections in future steps, and (iii) in which the qualities of consecutive chunks are non-increasing. This policy reduces the number of possible schedules per number of parallel connections from $Q^M$, where $M = n_e + |\mathcal{E}^b| - m$ is the number of remaining chunks, $m$ is the number of already requested chunks, and $Q$ is the number of possible quality levels for each remaining chunk, down to $\binom{M+Q-1}{Q-1}$.

As condition (iii) can still allow for a large number of candidate schedules, we also consider a more conservative policy that modifies condition (iii) so as to require the quality of the requested chunks to be sustainable for the remaining chunks. This policy, called *Optimized maintainable quality*, requires only $Q$ candidate schedules to be considered per number of parallel connections. Both policies pick the highest feasible quality for the next chunk, and open up additional parallel connections only if this quality can still be sustained.

To explore the tradeoff between aggressively picking high quality segments and prioritizing opening up additional parallel connections, we also consider two additional policies. Both these policies use the same three schedule constraints used with *Optimized maintainable quality.* In the first policy, called *Single connection*, we only allow a single connection to be open at each point in time. The second policy, called *Greedy bandwidth*, is much more bandwidth aggressive, and chooses chunk quality and the number of parallel new downloads so as to maximize the number of requested bytes $\sum_{i=j}^{j+m} q_i l_i$, when chunks $j$ through $j+m$ are requested at this time instance. Table 2 summarizes our policies.

## 3.4 Capped Workahead

A common on-off control strategy used by HAS players is to stop requesting chunks whenever the buffer occupancy exceeds a threshold $T_{max}$, until the buffer occupancy drops below another threshold $T_{min}$ [1, 3].

We generalize this idea to the context of branched video. Consider a player that is currently playing a segment $e \in \mathcal{E}$ and is approaching branch point $b$ with $|\mathcal{E}^b|$ path choices. We define a minimum buffer threshold $T_{min} = T_{single}|\mathcal{E}^b|$, where $T_{single}$ is the amount of data that should be buffered for each path choice to be confident that a stall will not occur, but such as to avoid excessive buffering along that path, in the case that path was not selected. Using the minimum buffer threshold, we then simply set the maximum buffer threshold $T_{max} = (T_{min} + \Delta)$, where $\Delta$ is an additional buffer margin.

## 4. IMPLEMENTATION DESCRIPTION

Our client is implemented using the Open Source Media Framework (OSMF v2.0) libraries and the front-end player is built from Strobe Media Playback (SMP). While most modifications are focused on the classes under the `http-streaming` directory, responsible for HAS, some modifications to the files under the `StrobeMediaPlayback` directory were needed to implement user interaction[2].

## 4.1 High-level System Overview

After having downloaded the HAS manifest file (that describes the HAS media file and the different encodings at which the chunks are available) and the metafile for the branched video (that describes the branched video structure), the client bootstraps the player using the information from the two files.

Our modified HAS player can open parallel TCP connections, and request and download multiple chunks in parallel. The chunks belonging to the segment currently being played back are delivered in-order to the playback buffer, whereas prefetched chunks from past an upcoming branch point are downloaded into the browser cache, from which they can be very quickly retrieved when the client makes a path decision. This design has been tested and compared against in-player memory solutions such as those presented by Krishnamoorthi et al. [17], showing significant reductions in in-player memory requirements and most importantly in the time that it takes to load a new path. This is important when masking the load times from cache to player, so as to ensure that users do not observe any playback interruptions.

Finally, to minimize fetch times from the cache, our current implementation uses the browser RAM cache (of our Mozilla Firefox version 25.0.1 browser), and requests from the player are checked against the index of cache content before placing a request to the server. While we have found benefits to using a RAM cache, relative to the use of a disk cache, these differences are negligible compared to the overall benefits of prefetching to cache in general.

[2]Our data files and source code for our software framework are available at `http://www.ida.liu.se/~nikca/papers/mm14.html`

Table 2: Summary of prefetching policies.

| Policy | Connections | Schedules considered | Objective |
|---|---|---|---|
| Optimized non-increasing quality | $1 \leq c_i \leq C^{max}$ | $\binom{M+Q-1}{Q-1}$, where $M = n_e + \lvert\mathcal{E}^b\rvert - m$ | Equation (2) |
| Optimized maintainable quality | $1 \leq c_i \leq C^{max}$ | $Q$ | Equation (2) |
| Single connection | 1 | $Q$ | Equation (2) |
| Greedy bandwidth | $1 \leq c_i \leq C^{max}$ | $Q$ | $\sum_{i=j}^{j+m} q_i l_i$ |

## 4.2 Interactive Playback

Once playback is close to a branch point, a message is displayed to the user requesting selection of the desired path option. The player reads the user input and this information is communicated to the class that handles branch-point transitions. The player also maintains a record of chunks and segments it has played previously. This information is used whenever the player has to use the longest matching rule (Section 2.2).

Once playback reaches a branch point, a branch-point event is triggered. Using the time associated with the branch point, the player calculates the chunk that should be played first following the branch point. Using this chunk number, our modified player checks if the chunk has been requested (at any quality level), and if so, determines its URL. Having determined the cached URL, the player can now quickly retrieve the content and move it into the playback buffer.

The likelihood of a stall event at a branch point is strongly impacted by the `BufferTime` parameter. This gives the minimum amount of data (in seconds) the player needs to have in the playback buffer before resuming playback after a stall or avoid a stall event at a branch-point event. Whenever our download manager triggers a branch-point event, we therefore set `BufferTime` to a small value so that there is no interruption in playback. As soon as playback is resumed after a branch point event, `BufferTime` is reset to the player's default value and additional requests are placed so as to fetch all the prefetched content for the new segment from the cache. The request module then resumes its normal operation and continues prefetching of the remaining chunks in the new segment and the first chunks following the next branch point.

## 4.3 Parallel Connections

An important aspect of allowing seamless playback is effective workahead prefetching. Researchers have found that HAS players often experience a large degradation in performance when exposed to competing flows [1, 13, 14]. Based on initial experiments and related literature (e.g., [4]) we conjecture that multiple parallel TCP connections will help ensure stable throughput in such scenarios.

We schedule new chunk downloads and open new TCP connections based on our workahead prefetching policies, as described in Section 3, which take into account playback deadlines, current download rate estimates, and the priority order of the chunks. Our policies are conservative, in that when calculating expected download completion times it is assumed that an additional TCP connection will not increase the total achieved download rate (and so would correspondingly reduce the rate achieved by each of the existing connections). At the end of each chunk download, new parallel connections are initiated only if this is not expected to lead to violations of the playback deadlines of chunks currently being downloaded or chunks yet to be downloaded.

## 4.4 Rate Estimation

Rate estimation is a critical component of any HAS player. Typically, rate estimates of the available bandwidth are generated based on the download rates observed during previous downloads. To avoid placing too much weight on the most recent chunk download, most HAS players make use of a weighted average of past download times/rates.

With our branched video player, we must also account for the fact that a chunk may have been downloaded in parallel with other chunks. For this reason, each new rate measurement is scaled up accordingly, using the conservative approximation that the number of such parallel connections was constant and equal to the minimum number of parallel connections during the chunk download. As with the default player, these measurements are then used to update an exponentially weighted moving average with weight 0.4.

## 5. EXPERIMENTAL RESULTS

## 5.1 Experimental Setup

To validate our general system design and evaluate the performance of different prefetching policies, we use a basic experimental testbed, in which a client and the server are connected over a high-speed LAN. To emulate a wide range of network conditions the client machine runs `dummynet` [23], allowing us to control the available bandwidth and round-trip-times (RTTs) observed between the two machines.

The server runs Flash Media Server (FMS) version 4.5, and hosts a HAS video encoded at four bit rates (250Kb/s, 500Kb/s, 850Kb/s, and 1300Kb/s), which we refer to as quality levels 0, 1, 2, and 3, respectively. The chunk size is 4 seconds. This HAS video was generated from the Big Buck Bunny video using Adobe's encoding and HAS packaging suite. Different numbers of segments and branch points were defined from the base video for different experiments.

The player is instrumented to continually write its internal state and actions to a log file, which we later process to evaluate the performance experienced by the client. Of particular interest is the buffer occupancy, the frequency and duration of stall events, as well as the general playback rate and quality experienced by the client.

The performance of our system design, is evaluated under each of the four prefetching policies *Optimized non-increasing quality*, *Optimized maintainable quality*, *Single connection*, and *Greedy bandwidth*, all defined in Section 3.3, as well as a *Naïve* player that is capable of handling interactive branched video, but that does not perform prefetching along the different alternative paths, but simply downloads chunks only along the current default path. Naturally, this policy stalls at each branch point that the default path is not selected and simply serves as a baseline for comparison.

By comparing the two optimized policies, *Optimized non-increasing quality* and *Optimized maintainable quality*, we can evaluate the value of being conservative, as the second policy should be considerably more conservative in its quality selection for the next segment. By comparing the performance of this policy with the *Single connection* policy, we can assess the value of using parallel connections. Finally, *Greedy bandwidth* helps further capture the tradeoffs associated with aggressively opening up parallel connections.

In the following subsections we present experiments in which we vary one parameter at a time. Starting from a default scenario, we give initial insights into the impact that factors such as the available bandwidth, end-to-end RTTs, number of chunks per segment, number of branches, and amount of competing traffic have on the playback performance. Results are presented both for the first branch point and for later branch points. For Sections 5.2-5.4 we use $T_{single} = 8$, but do not place any workahead (buffer size) limitations (Section 4.4). This limitation is instead separately validated in Section 5.5. The focus of the evaluation is on client playback performance (as measured by stall probability and playback rate[3]) and we only consider a single player (possibly competing against other TCP flows, but not other similar players). Interesting directions for future work include multi-factor experiments, experiments with competing players, and experiments with more dynamic bandwidth conditions (e.g., as seen by mobile users).

## 5.2 Single-branch Experiments

Consider first a scenario in which the client views a basic branched video (e.g., as shown in Figure 1) with a single initial segment $e$ with $n_e$ chunks, and a single branch point $b$, with $|\mathcal{E}^b|$ branch options. In our default scenario we let $n_e = 5$ and $|\mathcal{E}^b| = 4$. Furthermore, we use a default end-to-end bandwidth of 2,500Kb/s, RTT of 150ms, and no competing traffic. The use of an RTT of 150ms is motivated by typical values observed on our networks. For example, trace route measurements to the top million websites on the Web, according to `www.alexa.com` on June 12, 2012, suggests that a client within our campus would see an average RTT of 177.4ms to these sites.

To evaluate the impact that each of the above experiment parameters has on performance, we use a "one-factor-at-a-time" approach in which we run multiple series of experiments. In each series we vary one parameter at a time, and for each configuration in that series, we perform 30 experiments per policy, and report averages and standard deviations for the average video quality and stall probabilities.

In our experiments, we consider a worst-case scenario in which the client always picks the least likely branch, which is the last branch that our prefetching policies prefetch.

Figure 3 shows the video quality distribution for our default scenario, based on which we perform our one-factor experiments. Figure 4 shows the impact of the available bandwidth on the average video playback rate and stall probability. Here, the data points for 2500 Kb/s corresponds to our default scenario, and the average quality simply corresponds to the weighted sum of the quality distributions in Figure 3. As expected, the Naïve policy, which does not perform prefetching, always results in the most stall events. Clearly, the high playback rate of this policy is misleading when evaluating the playback quality. All of the prefetching policies successfully employ some of the bandwidth that the Naïve policy uses for downloading the current segment, for prefetching instead, so as to avoid stalls. The most successful tradeoff is achieved by the *Optimized maintainable quality* policy, which adapts its quality entirely based on our optimization formulation. In contrast to the *Optimized non-increasing quality* policy, this policy is conservative in that it does not schedule high quality downloads unless it expects this quality to be sustainable. The success of a more conservative policy is particularly evident when comparing against the *Greedy bandwidth* policy, which sees both higher stall probability and lower playback rate than the *Optimized maintainable quality* policy.

We next take a closer look at the impact of the end-to-end RTTs, number of chunks $n_e$ in the initial segment, and the number of branch options $|\mathcal{E}^b|$. Figures 5 and 6 show the playback rate and stall probability as a function of these variables. As expected, the average quality observed by the quality-aware policies increase with (i) decreasing RTTs, (ii) increasing number of chunks $n_e$ in the initial segment, and (iii) decreasing number of branch options $|\mathcal{E}^b|$. In the case of decreasing RTTs, the TCP throughput quickly ramps up. In the case of increasing number of chunks $n_e$, the policies have more time to build a workahead before reaching the branch point, and finally, in the case of decreasing number of branch options, less data must be downloaded ahead of the branch point, leaving bandwidth which instead can be used to download higher quality chunks.

As when comparing across different bandwidth conditions, the quality-aware policies consistently outperform the *Greedy bandwidth* policy. Under challenging conditions, the *Optimized maintainable quality* policy is the clear winner, and consistently sees the lowest stall probabilities.

## 5.3 Multiple Branch Point Scenarios

In general, we assume that the distance between successive branch points is long enough that it is never worthwhile to prefetch beyond more than a single branch point. We have also performed experiments in which the branched video structure has multiple branch points, to assess how the performance differs at the later branch points. In these experiments we have used symmetric structures in which all segments have the same number of chunks $n_e$ and all branch points have the same number of branch options $|\mathcal{E}^b|$.

Figures 7 and 8 show the average playback rate and stall probabilities for the later segments and branch points for experiments with a symmetric tree structure with depth three. While these results are consistent with those observed for the first branch point, we observe slightly higher overall playback rate for these later segments, especially for the prefetching policies with multiple parallel connections. This can be explained by the fact that these policies can continue to utilize any initiated parallel connections after passing the first branch point. However, in a few cases we also observe higher stall probabilities than for the first branch point. This is particularly apparent where there are only two chunks per segment. In this case, segments are too short to allow sufficient time for prefetching chunks from beyond the next branch point, before it is reached.

## 5.4 Competing Flows

We compare performance in scenarios with different numbers of competing TCP flows, and a total available bandwidth of 5Mb/s. Each competing flow was generated by downloading a large file from the same server.

[3]Startup delay is another important metric, but this is small in our experiments and does not differ among the policies.

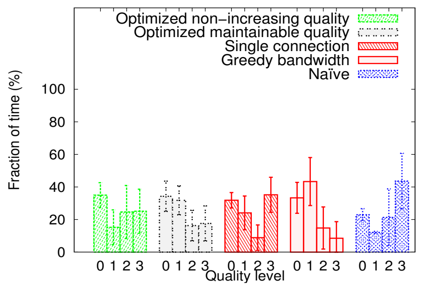
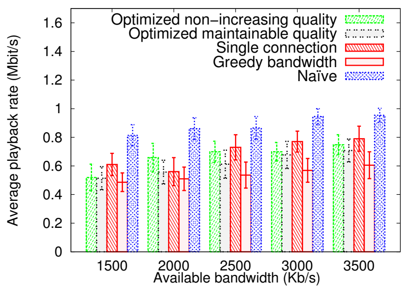
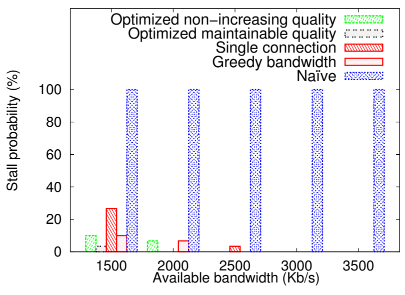


**Figure 3: Playback qualities in default scenario.**

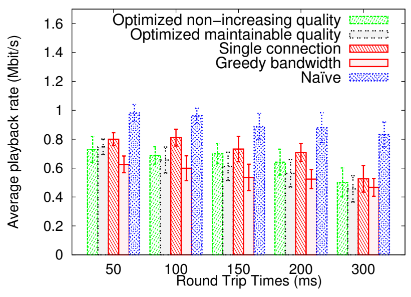


(a) Playback rate

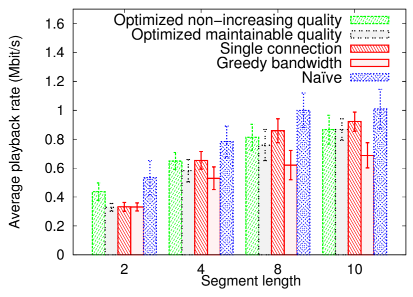


(b) Stall probability

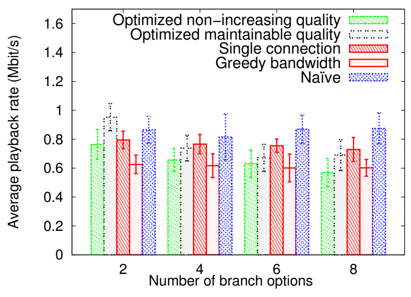
**Figure 4: Impact of the available bandwidth.**



(a) End-to-end RTTs



(b) Chunks $n_e$



(c) Branch options $|\mathcal{E}^b|$

**Figure 5: The average playback rate under different end-to-end RTTs, number of chunks $n_e$ in the initial segment, and the number of branch options $|\mathcal{E}^b|$.**

As seen from the results shown in Figure 9, there is a clear overall performance degradation as the number of competing flows increases. This is to be expected, as in these cases, the fair share of bandwidth available to the player quickly decreases, and the player must adapt the playback quality accordingly. However, in these cases, we can also see significant advantages to our prefetching policies that open multiple parallel connections, compared to the baseline *Single connection* policy.

## 5.5 Capped Workahead

The policies used in Sections 5.2-5.4 do not consider workahead (buffer size) limitations. While much of our analysis and that evaluation is focused on seamless playback, we have found that our capped workahead policy (Section 3.4) can save substantial bandwidth. To illustrate its operation, we include example results for a high-bandwidth scenario in which a client with 6Mb/s connection is playing a branched video with maximum rate encoding at 1.3Mb/s.

Figure 10 shows the variation in the buffer occupancy over time. In this scenario, we use the *Optimized maintainable quality* policy and the next upcoming branch point has 4 branch alternatives. With our default settings of $T_{single} = 8$ and $\Delta = 4$, this gives us $T_{min} = 32$ and $T_{max} = 36$.

It may appear unintuitive that we reach a buffer occupancy much higher than $T_{max}$. However, this can be explained by the use of many parallel connections. With a high available bandwidth, the player can be fairly aggressive in opening new parallel connections, especially initially when building up the initial buffer to $T_{max}$. With the buffer only accounting for completed in-order chunk downloads, the buffer occupancy can therefore easily overshoot. However, when reaching (and exceeding) $T_{max}$ some connections are terminated. With less parallel connections being able to accumulate during the later on periods, as we can see in the figure, the later peaks in buffer occupancy are significantly smaller than the original peak.

## 6. RELATED WORK

Much research on on-demand streaming of branched video, sometimes referred to as nonlinear media, has focused on server-side optimizations for popular content [7, 11, 27], including the development of multicast-based solutions [7,27]. Other works have designed authoring and media representation tools for the development of interactive media [21,25, 26], or tag-based systems that allow tagged parts of different videos to be automatically stitched together into a single playback sequence [16].

Recently, Krishnamoorthi et al. [17] proposed the idea of using HTTP-based Adaptive Streaming (HAS) together with branched video. HAS provides an excellent framework for delivering interactive branched video. While they provide a basic proof-of-concept implementation, the focus is on the general flexibility the framework enables, both for the content designer (of the interactive branched video) and the player (which can select to prefetch chunks of different qualities). This paper substantially extends this prior work by addressing the key policy question of how the player should decide what chunks to download when. This includes optimized policies, improved connection/data management, and a detailed evaluation. Meixner et al. [20] have also considered download and caching strategies for playback of branched video, but do not consider HAS.

Motivated by their widespread usage [10, 24], linear HAS players have been widely studied. For example, Akhskabi et al. [3] present an experimental evaluation of HAS players, and the impact their different quality adaption algorithms,

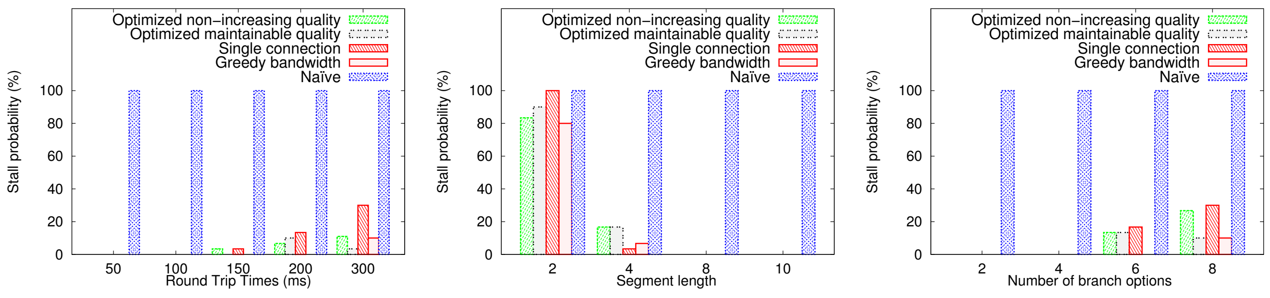

(a) End-to-end RTTs (b) Chunks $n_e$ (c) Branch options $|\mathcal{E}^b|$

**Figure 6: The stall probability under different end-to-end RTTs, number of chunks $n_e$ in the initial segment, and the number of branch options $|\mathcal{E}^b|$.**

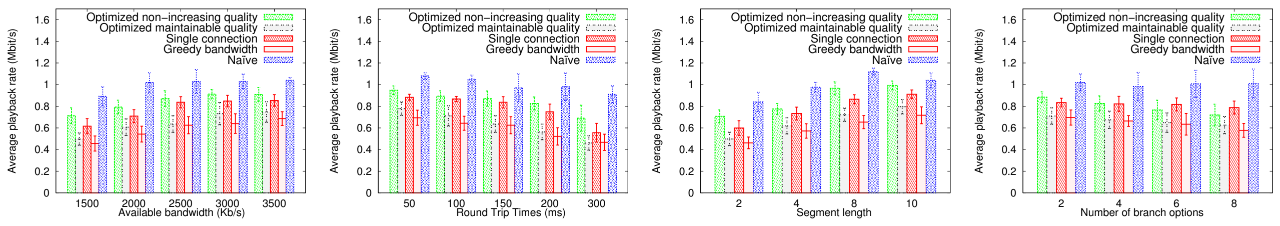

(a) Available bandwidth (b) End-to-end RTT (c) Chunks $n_e$ (d) Branch options $|\mathcal{E}^b|$

**Figure 7: Average playback rate in scenario with multiple branch points, under different conditions.**

buffer management policies, and other design choices have on the users' playback quality. Huang et al. present a detailed analysis on rate estimation at the HTTP layer and different factors that contribute to instability and unfairness in HAS players [13], and propose a buffer occupancy based solution for rate adaptation [14]. De Cicco et al. [8] propose an alternative buffer occupancy based solution, which generates a long-lived TCP flow instead of an on-off flow.

For the purpose of addressing oscillations and instability in quality choices made by the player, Akhshabi et al. [2] propose using a server-based traffic shaper, while Jiang et al. [15] develop an algorithm for chunk scheduling, quality selection and bit-rate estimation that improves fairness, efficiency and stability in HAS players. HAS performance is also influenced by player interaction with caches and other middle boxes [6, 18]. Both competing HAS traffic and regular TCP flows have an effect on the rate estimation and quality adaptation of the player [1, 13].

Other works have considered the impact of quality switching on the Quality of Experience (QoE) [19, 22], typically suggesting that QoE is improved with less quality switching. While our policies by design select rates such as to avoid future down switches in playback quality, we note that the least weighted branches (which are more likely to be obtained at a lower quality) are more likely to see a temporal quality degradation. Future work could investigate potential quality oscillations in the context of branched video.

## 7. CONCLUSIONS

Playback interruptions and stall times can greatly impact the users' playback experience. In this paper we present the implementation of a branched video streaming player that achieves seamless transition between segments in a branched video structure without playback interruptions. Using a simple optimization framework, we design optimized prefetching policies that maximize the playback quality while ensuring sufficient workahead to avoid stall events. To achieve these goals, our policies (i) adjust the quality levels of the prefetched chunks based on the bandwidth conditions that the client experiences, and (ii) determine the appropriate number of parallel connections to ensure good download speeds and improved workahead buffering. Our results show that our solution is able to achieve rate adaptive streaming and buffer management, such as to ensure seamless playback. We also show that the proposed solution effectively adapts the quality selection and number of parallel connections so as to provide the user with the best possible video quality under scenarios with different bandwidths, RTTs, segment lengths leading to a branch point, number of branch options after a branch point, and number of competing TCP flows. While our current system is entirely client-driven, future work includes server-side optimizations.

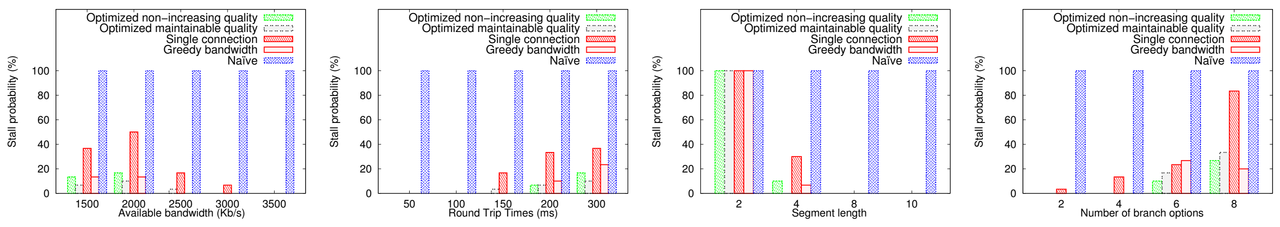
## 8. ACKNOWLEDGEMENTS

The authors are thankful to our shepherd Carsten Griwodz and the anonymous reviewers for their feedback, which helped improve the clarity of the paper. 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).

## 9. REFERENCES

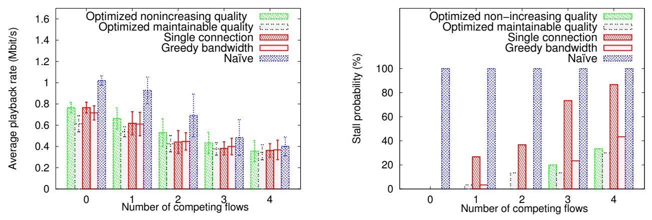
[1] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. What happens when HTTP adaptive streaming players compete for bandwidth? In *Proc. ACM NOSSDAV*, Feb. 2012.

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



(a) Available bandwidth (b) End-to-end RTT (c) Chunks $n_e$ (d) Branch options $|\mathcal{E}^b|$

**Figure 8: Stall probability at the second branch point under different conditions.**



(a) Playback rate (b) Stall probability

**Figure 9: The average video playback rate and stall events for different numbers of competing flows.**

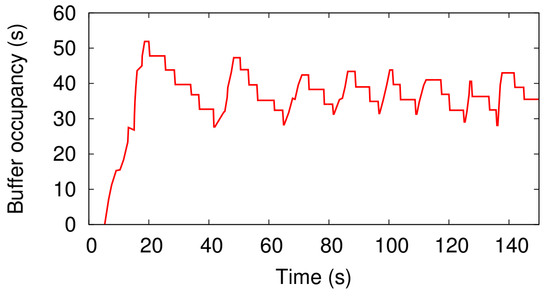


**Figure 10: Buffer occupancy over time**

[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*, Feb. 2011.

[4] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic. Parallel TCP sockets: Simple model, throughput and validation. In *Proc. IEEE INFOCOM*, Apr. 2006.

[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. Journal*, 16(2):101–114, Sep. 2011.

[7] N. Carlsson, A. Mahanti, Z. Li, and D. L. Eager. Optimized periodic broadcast of nonlinear media. *IEEE Trans. on Multimedia*, 10(5):871–884, 2008.

[8] 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*, Dec. 2013.

[9] 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*, Aug. 2011.

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

[11] D. Gotz. Scalable and adaptive streaming for non-linear media. In *Proc. ACM MM*, Oct. 2006.

[12] C. Griwodz, F. T. Johnsen, S. Rekkedal, and P. Halvorsen. Caching of interactive multiple choice MPEG-4 presentations. In *Proc. IEEE IPCCC*, Apr. 2006.

[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*, Nov. 2012.

[14] T. Huang, R. Johari, and N. McKeown. Downton abbey without the hiccups: Buffer-based rate adaptation for HTTP video streaming. In *Proc. ACM SIGCOMM FhMN Workshop*, Aug. 2013.

[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*, Dec. 2012.

[16] D. Johansen, P. Halvorsen, H. D. 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. *Multimedia Tools and Applications*, 61:419–445, 2012.

[17] V. Krishnamoorthi, P. Bergström, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. Empowering the creative user: Personalized HTTP-based adaptive streaming of multi-path nonlinear video. In *Proc. ACM SIGCOMM FhMN Workshop*, Aug. 2013 (Also ACM SIGCOMM CCR, 43(4):591-596, Oct. 2013.).

[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*, Aug. 2013.

[19] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. Probe and adapt: Rate adaptation for http video streaming at scale. *IEEE Journal on Selected Areas in Communications*, 2014.

[20] B. Meixner and J. Hoffmann. Intelligent download and cache management for interactive non-linear video. *Multimedia Tools and Applications*, pages 1–44, June. 2012.

[21] B. Meixner and H. Kosch. Interactive non-linear video: Definition and XML structure. In *Proc. ACM DocEng*, Sep. 2012.

[22] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang. Qdash: A QoE-aware DASH system. In *Proc. MMSys*, Feb. 2012.

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

[24] Sandvine. Global Internet phenomena report- 2H 2013, Technical report, Nov. 2013.

[25] A. Sobe, L. Böszörmenyi, and M. Taschwer. Video Notation (ViNo): A Formalism for Describing and Evaluating Non-sequential Multimedia Access. *International Journal on Advances in Software*, 3(1 and 2):19–30, Sep 2010.

[26] U. Spierling, S. A. Weiß, and W. Müller. Towards accessible authoring tools for interactive storytelling. In *Proc. ACM TIDSE*, Dec. 2006.

[27] Y. Zhao, D. L. Eager, and M. K. Vernon. Scalable on-demand streaming of nonlinear media. *IEEE/ACM Trans. on Networking*, 15(5):1149–1162, Oct 2007.