Optimized Adaptive Streaming of Multi-video Stream Bundles

Niklas Carlsson[†] Derek Eager[§] Vengatanathan Krishnamoorthi[†] Tatiana Polishchuk[†]
[†] Linköping University, Sweden
[§] University of Saskatchewan, Canada

***Abstract*—In contrast to traditional video, *multi-view* video streaming allows viewers to interactively switch among multiple perspectives provided by different cameras. One approach to achieving such a service is to encode the video from all of the cameras into a single stream, but this has the disadvantage that only a portion of the received video data will be used, namely that required for the selected view at each point in time. In this paper we introduce the concept of a "multi-video stream bundle" that consists of multiple parallel video streams that are synchronized in time, each providing the video from a different camera capturing the same event or movie. For delivery we leverage the adaptive features and time-based chunking of HTTP-based Adaptive Streaming (HAS), but now employing adaptation in both content and rate. Users are able to change their viewpoint on-demand and the client player adapts the rate at which data is retrieved from each stream based on the user's current view, the probabilities of switching to other views, and the user's current bandwidth conditions. A crucial component of such a system is the prefetching policy. For this we present an optimization model as well as a simpler heuristic that can balance the playback quality and the probability of playback interruptions. After analytically and numerically characterizing the optimal solution, we present a prototype implementation and sample results. Our prefetching and buffer management solution is shown to provide close to seamless playback switching when there is sufficient bandwidth to prefetch the parallel streams.**

I. Introduction

The average connection bandwidth for Internet users has increased by more than an order of magnitude per decade over the past few decades and is expected to continue improving at a similar rate. These advances have enabled companies such as Netflix to offer on-demand video streaming services that are challenging the traditional TV providers, and are expected to enable many additional innovative video streaming applications that will deliver us tomorrow's entertainment.

In particular, there has been considerable recent interest in video streaming in which users are able to interactively choose among the views provided by different cameras. Consider for example a video recording of a sporting event. Usually multiple cameras are used in capturing the event, but selection among these cameras is controlled by the producer, so that at any particular point in time video from only one camera is being included in the recording. It would be preferable, however, if the views provided by all cameras were available to the viewer, with each viewer able to interactively switch among these as desired during playout without playback interruption. Other use cases include reality TV shows such as "big brother", crowd-sourced recordings of a concert, and movies produced with multiple cameras so as to allow multiple perspectives.

One way to provide multi-view streaming is to encode the video from all views into a single stream [1], [2], [3], [4]. However, this approach has the disadvantage that only a portion of the received video data will be used, implying lower achievable video quality given fixed client bandwidth. Instead, we propose here an approach in which multiple (regular) video streams, capturing the same scenes from different perspectives, are used to create what we term a "multi-video stream bundle". Crucially, each client does not have to receive all of this content at the same desired playback video quality, but instead the data (and qualities) retrieved depend on the viewer's current viewpoint selection and viewpoint switching probabilities. Viewers are able to dynamically switch among the streams in the bundle, at arbitrary times, while being provided with seamless playback at a high playback quality.

To provide such a service, in addition to streaming the currently played stream, the player must be able to (i) adaptively prefetch alternative streams that the user may switch to, at quality levels dependent on the switching likelihoods, and (ii) seamlessly (during playback) stitch together the playback sequences of multiple videos when a user selects to switch to a different stream. Adaptive prefetching is needed to ensure that the available bandwidth is best used to prepare the player for potential future stream switching and careful buffer management is needed to ensure that clients do not endure noticeable playback interruptions when switching streams. Current players do not provide these functionalities.

In this paper we present a general multi-video stream bundle framework that leverages the quality adaptive features and natural time-based chunking of modern HTTP-based Adaptive Streaming (HAS) systems. The use of HAS allows us to adapt the streaming quality of each stream based on the user's current bandwidth conditions and estimated stream switching probabilities, so as to best balance the playback quality and the probability of playback interruptions at the time of stream switching. The paper makes three major contributions.

First, we present the design of a novel multi-video stream bundle system (Section II). Our design introduces the concept of multi-video stream bundles (defined above), allows content producers to easily define default stream(s), and gives users the freedom to switch between alternative viewpoints at arbitrary times. Our design combines dynamic bandwidth sharing, which determines the share of the bandwidth given to the currently played video stream versus that given to the alternative streams that the user may switch to, with a chunk quality selection policy for the alternative streams.

Most recordings are done with conventional cameras. Motivated by this observation, we assume a simple service in which users can only view one (conventional) video stream at a time. Although some of the results and insights provided in this paper might be extendable to free viewpoint (where videos from multiple strategically placed cameras can be combined so to create virtual viewpoints [5], [6]) and tiled video streaming (where multiple tiles of an often panoramic or omni-directional view can be combined to allow panning, tilting, and zooming functionalities [7], [8], [9]), extensions for services in which the user's viewpoint is a combination of multiple parallel video streams are left as future work.

Although there are recent works on multi-view video streaming [3], [4] and tiled streaming [10], [9] that have used a HAS-based design, to the best of our knowledge, our framework is the first to perform quality adaptation of a set of conventional video-on-demands streams based on *both* current bandwidth conditions and switching probabilities. In contrast to free viewpoint videos [6], the video streams in a stream bundle can be individual recordings and require no depth information. In contrast to free viewpoint videos and region-of-interest (ROI) based streaming using tiled videos (which typically require specialized high definition panoramic or omni-directional cameras) stream bundles can be composed of streams from any set of cameras. The concept of stream bundles also has the advantage that the bandwidth consumed by streams can easily be adjusted based on their popularities. This is particularly important for content providers that would make large bundles available. In contrast, many multi-view video streaming solutions encode all views (popular and less popular) into a single stream [1], [2], [3], [4].

Second, we provide analytically (Section III) and numerically (Section IV) derived insights into the characteristics of optimized prefetching policies for alternative streams. We present an optimization framework to determine the download schedule of the optimized prefetching policies. The framework addresses the problem of chunk quality selection for the alternative streams and balances considerations of playback quality and probability of playback interruptions, given current network bandwidth constraints and stream switching probabilities. Using the framework, we then derive and identify properties of the optimal solution for our specific problem, and provide insights into how the generally NP-hard mixed integer linear programming (MILP) formulation can be solved efficiently for various special cases. The optimal solution and the accuracy of a simpler heuristic are analytically (Section III) and numerically (Section IV) characterized for example systems.

Finally, we present a prototype implementation (Section V) of our multi-video stream bundle system and show that it is able to achieve effective prefetching and close to seamless stream switching when there is sufficient bandwidth for prefetching alternative streams. Using our system implementation, we demonstrate the feasibility of our solution and show that our solution provides significant performance improvements over alternative baseline implementations.

The remainder of the paper is organized as follows. Section II presents the overall system design. Sections III and IV present our chunk quality selection optimization framework and a numerical characterization of the optimal solution, respectively. Our implementation and evaluation thereof are presented in Section V. Finally, related work (Section VI) and conclusions (Section VII) are presented.

II. Multi-video Stream Bundling

At the core of our system design is the concept of a "multi-video stream bundle" that consists of multiple "parallel" video streams that are synchronized in time, each providing the video from a different camera capturing the same event or movie. In contrast to prior work on multi-view streaming, we employ a HAS-based approach, in which each individual stream of a stream bundle is individually encoded and delivered using HTTP-based Adaptive Streaming (HAS). Leveraging that each stream is split and encoded into chunks of different qualities, we provide bandwidth-aware prefetching (involving quality adaptation across both time and individual streams) and carefully split the bandwidth allocated to the currently played stream and alternative streams that the user may choose to switch to next. Our solution is client-driven and does not depend on whether the service is provided in the cloud, by CDN servers, or a single origin server, for example.

A. *Streaming Structure*

A stream bundle can easily be created by leveraging parallel video recordings (professional or crowd-sourced) from any sporting event or concert, or from parallel recordings typically done when recording a movie or TV show, for example.

Interactive personalized streaming: With our design, each user can change which stream they are viewing at any point in time. By allowing users to instantaneously and repeatedly change which stream they view, our stream bundle design enables interactive personalized streaming experiences.

Creator defined vs user-driven paths: A content creator can easily define a "default path", if desired, but also allow the users to select their own alternative paths through the stream bundle. We define a path as a sequence of stream switches. In the simplest scenario, the default path consists of the "video cuts" selected by a professional editor and the other streams contain parallel recordings from alternative viewpoints.

B. *Adaptive Streaming, Prefetching, and Seamless Switching*

Bandwidth-aware prefetching: The use of HAS allows for quality adaptation across *both* time and individual streams. With HAS, a video stream is encoded into chunks that are of equal play duration (e.g., 2-5 second long), each available in different quality encodings, and encoded so they can be played individually. The use of multiple encodings per chunk and aligned chunk boundaries allow the client player to adaptively choose the most suitable encoded chunks of that stream based on the current conditions. In addition to downloading chunks for the currently played stream, in our context, bandwidth-aware prefetching is also used to download chunks for the parallel streams that the user may be most likely to switch to.

Quality maximization and seamless stream switching: The most important parts of our solution are the dynamic bandwidth sharing (Section II-C) and the prefetching policies (Section III) that determine which chunks of each stream should be downloaded, and the quality for each of these chunks. Ideally, such policies should make best possible use of the available bandwidth, adapt to current conditions, and provide seamless playback at the highest possible quality.

Prefetching policies are needed that determine the set of chunks and chunk qualities that best balance the goals of maximizing the playback quality and minimizing the probability of playback stalls at switching instances. We define an optimization framework for such policies, derive optimal policies, and characterize their structure and computational complexity. We then derive a simpler heuristic, evaluate its performance relative to the theoretic optimal, implement it in a real system, and show that it works in practice.

Whereas precise switching probabilities may be difficult to assign for individual users, we note that there are many scenarios where the relative order of these probabilities could be easily predicted. For example, in many cases physical constraints may cause some camera views to be relatively closer to each other than others. In these cases, users may be more likely to switch to such neighboring streams. Streams may also have a natural viewing popularity ordering according to their coverage of some central element of a scene (e.g., the main actor, or the area of current action in a sporting event).

Buffer management: In addition to prefetching policies, our proof-of-concept implementation includes careful buffer management to allow caching of alternative streams and seamless switching of video containers at the times that a user chooses to switch video streams during ongoing playback. The chunks belonging to the currently played stream are delivered in-order to the playback buffer, whereas prefetched chunks from parallel streams are downloaded into the browser cache, from which they can be quickly retrieved when the user switches streams. At the time of a stream switch, the playback buffer is flushed and a new video container is quickly loaded.

C. Dynamic Bandwidth Sharing

HAS players must handle time-varying buffer conditions. For example, even though HAS players typically spend most time in steady state conditions [11], a player starting with an empty buffer must go through a transient phase during which the buffer is filled. In this section, we describe how our system determines the share of the bandwidth given to the currently played stream versus that given to the alternative streams.

1) Determining Bandwidth Share: To allow for smooth streaming it has been suggested that players should use the current buffer occupancy for determining their quality adaptation choices [11]. Inspired by the buffer-based reservoir concept proposed by Huang et al. [11] and implemented in (proprietary) Netflix players, we keep track of both the amount of buffered data T (measured in seconds) of the currently played video stream and an estimate of the total available bandwidth capacity C_{est} (measured in bytes per second). Using this information, the bandwidth capacity is then split into two parts: (i) C_{play} is used for the quality selection when downloading chunks of the currently played video stream, and (ii) $C_{pref} = C_{est} - C_{play}$ is used for prefetching chunks of

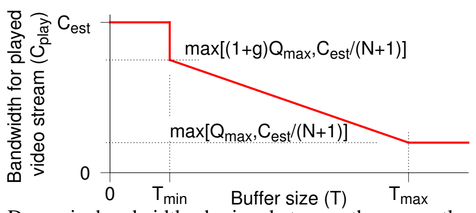


Fig. 1. Dynamic bandwidth sharing between the currently played video stream and the prefetching of parallel video streams.

the other parallel streams. Our objective is to bias this split so that we prioritize the current stream (C_{play}) when the amount of buffered data T is low (so as to avoid stalls) and to give more bandwidth to prefetching (C_{pref}) when T is large.

Primarily designing for high-bandwidth scenarios where it is possible to download the currently played stream at highest possible quality while also prefetching data for the alternative streams, we assume that (on average) $C_{est} \geq (1+g)Q_{max}$, where $g \geq 0$ is a parameter of the bandwidth sharing protocol and Q_{max} (measured in bytes per second) is the maximum quality encoding that the player may choose. For scenarios in which the maximum available quality encoding does not satisfy this condition, we simply constrain the client player to a Q_{max} value that does satisfy the condition.

Figure 1 illustrates the dynamic bandwidth sharing split, showing C_{play} as a function of the amount of buffered data T. Similar to the thresholds used by typical HAS systems, we use T_{min} and T_{max} for low and high buffer thresholds.

Under low buffer conditions (when $T < T_{min}$) the currently played video stream is given 100% of the estimated bandwidth capacity C_{est}. This facilitates building and maintaining a sufficient buffer of video data for the currently played stream. Between T_{min} and T_{max} the currently played stream is given a bandwidth share that follows a linear decreasing function, reflecting the decreasing need to fetch data for this stream at high rate. For $T > T_{max}$, the currently played stream is given a bandwidth share Q_{max}, unless there is excess bandwidth ($C_{est} > (N+1)Q_{max}$), in which case the bandwidth is split equally across the played stream and the N alternative streams, possibly allowing workahead on all streams. More formally:

$$C_{play} = \begin{cases} C_{est}, & \text{if } T \leq T_{min}, \\ (1-x)\max[(1+g)Q_{max}, \frac{C_{est}}{N+1}] & \\ \quad + x\max[Q_{max}, \frac{C_{est}}{N+1}], & \text{if } T_{min} < T < T_{max}, \\ \max[Q_{max}, \frac{C_{est}}{N+1}], & \text{if } T_{max} \leq T. \end{cases} \tag{1}$$

where $x = \frac{T - T_{min}}{T_{max} - T_{min}}$. Note that the condition $C_{est} \geq (1+g)Q_{max}$ ensures that additional buffer buildup (workahead) of the played video is always possible whenever $T < T_{max}$.

2) Chunk Download Schedule: Having determined the bandwidth share for both the played stream and for prefetching the alternative streams, we next describe how the chunk download schedule is determined during time-varying buffer conditions. The key idea here is to decouple the calculations of the bandwidth share and the choice of the next chunk to download. In particular, at the time a new chunk request is about to be made, we re-calculate the bandwidth shares C_{play} and $C_{pref} = (C_{est} - C_{play})$, as per the above calculations.

Using C_{pref} as the bandwidth allocated to prefetching we then use a quality selection policy (for which an optimization framework is defined in Section III) to determine which chunk quality q_i should be used for each stream i in the bundle. Figure 2 shows an example of a quality assignment structure that we later (in Section III-C) show is optimal under some circumstances. In the case there is not enough bandwidth for all streams, the algorithm also calculates the fraction f_i of chunks from stream i that should be downloaded.[1] The example of Figure 2 is for the special case when either $f_i = 1$ (when $1 \leq i \leq k$) or $f_i = 0$ (when $k < i$).

Finally, to pick the chunk to download next, round-robin ordering is used across the streams from which chunks will be downloaded, starting with the currently played video stream and followed by the alternative streams ordered according to an assigned stream weight that can reflect the relative probability of switching to that stream. During each round exactly one chunk from the currently played video stream is downloaded, as well as at most one chunk from each alternative stream.

Note that round-robin schedules can be non-optimal in some scenarios. For example, in scenarios where there is a high probability of switching away from the currently played stream to some particular alternative stream, it may be appropriate to not download at all from the currently played stream and/or to download more than one chunk from the alternative stream. However, we do not consider such scenarios here.

Figure 3 illustrates the round-robin scheduling approach. The timings of when chunks of the played stream are downloaded and played are all shown in red; so is the buffer level of the played stream. The download timings of prefetching streams are shown in green, blue, and black. Here we have assumed a stream bundle with at least four streams, in which the client is currently viewing the first and performs prefetching when above the minimum buffer threshold T_{min}. We also assume that the client uses a single TCP connection and only show the buffer of the played stream. In this example, the client initially downloads chunk 1 and 2 of the played stream at a low quality (these downloads are depicted by the small red rectangles, numbered 1 and 2, in the figure) before downloading later chunks of the played stream at a high quality (larger red rectangles, numbered 3, 4, 5 and 6). As dependent on the bandwidth capacity estimate and the amount of buffered data for the played stream (shown at the bottom of the figure), the client also prefetches low quality chunks of 2-3 alternative streams in round robin (small green/blue/black rectangles, each illustrated using a different line type). We note that all of the depicted downloads are completed well ahead of the respective playback deadlines (the playbacks of the first four chunks from the played stream are depicted by the rectangles in the top row of the figure), and that, as desired, the amount of prefetching (download of chunks from the alternative streams) is adapted based on the buffer conditions.

Although our discussion assumes a single TCP connection, the use of round-robin easily generalizes to multiple connections. In fact, we have validated our implementation using experiments with both a single connection and multiple parallel connections. However, similar to most other HAS work we focus here on the case of just a single TCP connection.

[1]For example, in the case $f_i = 0.5$, the client would download 50% of the chunks of stream i. Note that chunks are independently encoded in HAS/DASH systems. Downloading a fraction f_i of a stream's chunks would correspondingly reduce the stall probability when switching to that stream.

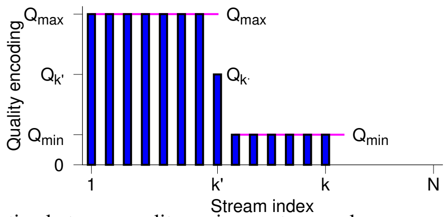


Fig. 2. Optimal stream quality assignment example.

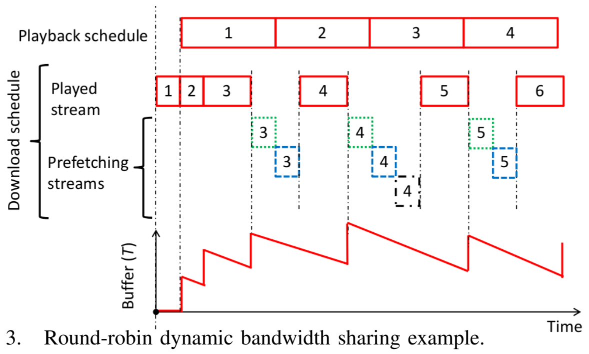


Fig. 3. Round-robin dynamic bandwidth sharing example.

For the currently played stream, C_{play} is used when selecting the encoding rate of the next downloaded chunk, exactly as the total download rate would be used with a regular player. Naturally, many optimizations and metrics can be taken into account here, including ones that try to reduce playback quality variations. Here, we just used the default player selection rule. For the other streams, we use the chunk encodings assigned by the quality selection policy based on the remaining capacity C_{pref}. Assuming back-to-back downloads at rate C_{est}, each download during a round should complete within the time it takes to play out a chunk of the currently played stream.

III. Optimization Model

In this section we develop an analysis and optimization framework for the problem of determining what chunks, at what qualities, should be prefetched. Our framework generalizes the buffer-based approach used by Huang et al. [11] to the stream bundle scenario. In particular, we consider the steady-state case in which new chunks are being added to the buffer for each alternative candidate stream i at the same average rate that they are being removed due to playback deadlines expiring. When the overall system is not in steady state, our analysis can be applied a single round-robin round at a time, for which the current estimate of C_{pref} allows determination of a prefetch schedule for that round.

We consider a single client streaming a multi-video stream bundle with $N+1$ parallel video streams. One of these streams is currently being played and the other N are the alternative streams, from which data will be prefetched (the "prefetching streams"). The client has bandwidth capacity $C = C_{pref}$ for the prefetching streams. Each video is chunked and available in n video qualities (encoding rates). For each chunk to be

TABLE I. SUMMARY OF NOTATION

Notation	Definition
C	Available bandwidth capacity for prefetching
N	Number of prefetching streams
$\mathcal{Q}$	Set of video qualities
n	Number of quality encodings ($n = \|\mathcal{Q}\|$)
Q_{min}	Minimum video quality encoding
Q_{max}	Maximum video quality encoding
w_i	Weight of stream i
q_i	Selected quality encoding of stream i
f_i	Selected fraction of chunks of stream i
$u_{i,j}$	Utility of stream i using choice j
$q_{i,j}$	Quality encoding of stream i using choice j
$b_{i,j}$	Average bandwidth of stream i using choice j
$x_{i,j}$	Binary assignment variable for stream i and choice j
A	Normalized stall penalty
k	Number of prefetching streams being downloaded from

downloaded from a stream i, the client must select at which quality encoding $q_i \in \mathcal{Q}$ it should download that chunk, where $\mathcal{Q}$ is the set of available qualities. We consider the case where $N > \lfloor \frac{C}{Q_{max}} \rfloor$ where $Q_{max} = \max q \in \mathcal{Q}$; otherwise, the client can simply select the maximum quality for all of the streams.

A. General Optimization Problem

In the following, we present a quite general analytic optimization model. As is standard with such models, we employ general concepts such as weights and utility functions. However, we also provide examples of how these could be more concretely defined. Table I summarizes our notation.

First, we let w_i be the normalized weight given to stream i, such that $\sum_{i=1}^{N} w_i = 1$, with the weights reflecting the relative prefetching priorities. Although these weights can be interpreted quite generally, for the purpose of simplifying our discussion we will consider the particular case in which the weight w_i given to each stream i is proportional to the probability of switching to stream i prior to the play time of a prefetched chunk for that stream. Without loss of generality, streams are ordered such that $w_1 \geq w_2 \geq ... \geq w_N$.

In the stream bundle scenario, the client player must balance the importance of receiving high quality chunks and the probability of playback stalls when switching streams, given fixed prefetching bandwidth capacity C. For each stream i, the player must therefore decide both (i) the fraction f_i of chunks that are downloaded, and (ii) at what quality encoding q_i these chunks are downloaded. Using these two factors, the average bandwidth usage is calculated as $b_i = f_i q_i$, measured as data per time unit (e.g., bytes/second). We assume a given mapping between the combination of these two factors and the estimated utility $u(f_i, q_i)$, when switching to stream i at a random time instance. Our optimization problem is to maximize the weighted client utility subject to the bandwidth constraint; i.e.,

$$\text{maximize} \sum_{i=1}^{N} w_i u(f_i, q_i), \ \text{subject to} \sum_{i=1}^{N} b_i \leq C. \qquad (2)$$

Although this is a concise formulation, the infinite solution space makes this problem hard to solve for general utility functions and arbitrary f_i values. Consider instead some discrete set of potential f_i values for each stream i, and enumerate the combinations with the potential q_i values. Let $b_{i,j}$ and $u_{i,j}$ denote the average bandwidth usage and utility for the j'th such combination. Introducing the binary variable $x_{i,j}$ for each i and j, which is 1 when combination j is chosen for stream i and 0 otherwise, the optimization problem can now be written as a mixed integer linear program (MILP):

$$\text{maximize} \sum_{i=1}^{N} w_i \sum_{j} x_{i,j} u_{i,j}, \qquad (3)$$

$$\text{subject to} \sum_{i=1}^{N} \sum_{j} x_{i,j} b_{i,j} \leq C, \qquad (4)$$

$$\sum_{j} x_{i,j} = 1, \quad x_{i,j} \in \{0, 1\}, \quad 1 \leq i \leq N, \quad \forall j. \qquad (5)$$

For the general case, this problem is NP-hard. To see this, note that the 0-1 knapsack problem (which is NP-hard) can be reduced to a special case of this problem in which all the w_i are 1 and there are only two choices for each stream (either download with probability 1 at the single available quality level for that stream, or do not download from the stream at all). The objective in this case is to maximize the sum over the utilities of the downloaded chunks, under the constraint that these downloads must fit within the bandwidth capacity.

Although this problem is NP-hard and the number of candidate solutions to consider can be exponential, moderately-sized problems can be solved using modern solvers. Here, however, we explore structure in the problem that can reduce the number of possibilities that need be considered. In the process we also provide insights into the characteristics of the optimal policies.

First, clearly we need to consider only combinations for each stream i such that combinations with higher bandwidth usage also have higher utility. Suppose that there are s combinations remaining for each stream after doing this pruning, and enumerate these choices based on increasing bandwidth usage $b_{i,j} = f_{i,j} q_{i,j}$ and utility $u_{i,j} = u(f_{i,j}, q_{i,j})$. Assuming the same choices across all streams, it is then easy to show that we need consider further only $\binom{N+s}{s}$ possible choices for the set of $x_{i,j}$ values. We first prove the following lemma.

Lemma 1: There exists an optimal solution for which $b_l \geq b_k$ whenever $l < k$.

Proof: (Lemma 1) Assume that there exists an optimal solution for which this property does not hold; i.e., there exists at least some $l < k$ for which $b_l < b_k$. Consider now an alternative solution in which the bandwidths used by streams l and k are exchanged such that $b'_l = b_k > b'_k = b_l$. This alternative solution is feasible since it consumes the same total bandwidth. Furthermore, due to the ordering of the streams, the objective function of the alternative solution is no less than the objective function for the original solution; i.e., $w_l u(b_l) + w_k u(b_k) \leq w_l u(b_k) + w_k u(b_l) = w_l u(b'_l) + w_k u(b'_k)$. Such exchanges can be repeated on a pairwise basis until all streams satisfy the condition given in the lemma statement. ■

This lemma shows that there is an optimal allocation of bandwidth that is non-increasing. The number of choices for the set of $x_{i,j}$ values that would need to be considered is therefore no greater than the number of unique monotone paths in an $N \times s$ grid, equal to $\binom{N+s}{s}$.

B. Baseline Scenario

We now take a closer look at the case where, for each stream i, the client either downloads a chunk for that stream in each round, or never does. In this case f_i is either 1 or 0, and can be interpreted as an indicator variable. Furthermore, to balance the objective of maximizing expected video quality and minimizing the probability of stalls we assume that each client endures a potentially client-dependent stall penalty (in the utility) of $-A$ whenever there is a stall event, and the client's utility when switching to stream i is otherwise proportional to the playback quality encoding q_i. In this case, the utility can be taken as $q_i \in \mathcal{Q}$ whenever $f_i = 1$, and $-A$ otherwise. Assuming $n = |\mathcal{Q}|$ quality levels, for this case, we can now write the optimization problem as

$$\text{maximize} \quad \sum_{i=1}^{N} w_i \left(\sum_{j=1}^{n} q_{i,j} x_{i,j} - A(1 - f_i) \right), \tag{6}$$

$$\text{subject to} \sum_{i=1}^{N} \sum_{j=1}^{n} q_{i,j} x_{i,j} \leq C, \tag{7}$$

$$\sum_{j=1}^{n} x_{i,j} = f_i, \qquad 1 \leq i \leq N, \tag{8}$$

$$x_{i,j} \in \{0,1\}, \quad f_i \in \{0,1\}, \quad 1 \leq i \leq N, 1 \leq j \leq n. \tag{9}$$

Here, $x_{i,j}$ is a binary variable equal to 1 when the j^{th} quality choice $q_{i,j}$ is assigned to the i^{th} stream, and 0 otherwise. Note that constraints (8) and (9) ensure that at most one term within the large brackets of the objective function (6) is non-zero at a time, giving the expected utility when switching to that stream.

Determining the best metrics for QoE in a new context (multi-video stream bundles), is a non-trivial and substantial problem in its own right. However, linear models weighting playback quality and stalls have been used in other contexts. For example, Yin et al. [12] use the linear combination of the average video quality, the average quality variation, the rebuffering times, and the startup delay to capture the quality of experience for linear video. Here, we instead focus on the presence (or absence) of stalls, rather than their duration, and do not model quality variations. While future work could incorporate a quality variation metric into the model, we believe that in our stream switching context, avoiding a stall after a stream switch would be more important to users than avoiding some brief quality variation after a stream switch. In Section V-E we compare the quality variations of our solution and two alternative baseline implementations.

While Model Predictive Control (MPC) and other control-based approaches provide promising future extensions, we believe that studying the simpler one-time shot optimization problem is an important first step that provides useful insight into the characteristics of optimal solutions for given available bandwidth. Such solutions are a desirable target for periods with stable available bandwidth. In some streaming contexts, at least, these stable periods appear to be quite common [13].

C. Special Cases

Insights can be gained into the optimal solution for our baseline scenario by considering some special cases.

Infinitesimal Granularity: Consider the limiting scenario when there is an unbounded number of quality levels ($n \to \infty$), with the difference between each pair of successive quality levels in the range $Q_{min} = \min q \in \mathcal{Q}$ to $Q_{max} = \max q \in \mathcal{Q}$ infinitesimally small. For this scenario, consider first the case when there is no stall penalty ($A = 0$) and the user is only interested in maximizing the weighted playback quality $\sum_i w_i f_i q_i$. For this case, the optimal policy is to greedily allocate as much bandwidth (and quality) as possible to the most weighted streams. Therefore, the optimal policy downloads from exactly $k = \lceil \frac{C}{Q_{max}} \rceil$ streams, with the following quality selections:

$$q_i = \begin{cases} Q_{max}, & 1 \leq i \leq k-1, \\ C - (k-1)Q_{max} & i = k. \end{cases} \tag{10}$$

Greedy exchange arguments can be used to show that this policy is optimal for the $A = 0$ case. Note that any alternative policy would need to select lower quality for at least one of the first $k-1$ streams with higher weights, and higher quality for at least one of the streams with lower weights, than what is assigned with the above policy. Such an alternative policy is sub-optimal since the weighted client utility would improve by moving some bandwidth from the lower weighted stream to increase the quality of a higher weighted stream. With infinitessimal granularity, this redistribution is always possible.

As the penalty $A > 0$ increases, the optimal policy will download from more and more streams, but has an easily defined structure. Suppose that k streams are being downloaded from in an optimal solution, for some k between $\lceil \frac{C}{Q_{max}} \rceil$ and $\min[N, \lfloor \frac{C}{Q_{min}} \rfloor]$. Then $k' - 1$ streams, where $k' = \lfloor \frac{C - kQ_{min}}{Q_{max} - Q_{min}} \rfloor \leq k$, will have the highest quality, $k - k'$ streams the lowest quality, and one stream will have a quality equal to one of, or between, these two extremes. This assignment is illustrated in Figure 2, and defined as follows:

$$q_i = \begin{cases} Q_{max}, & 1 \leq i \leq k'-1, \\ C - (k'-1)Q_{max} - (k-k')Q_{min}, & i = k', \\ Q_{min} & k' < i \leq k. \end{cases} \tag{11}$$

Using the same exchange arguments as used for the $A = 0$ case, note that this assignment cannot be improved by reallocating bandwidth among the streams being downloaded from.

Clearly, for the case when $A \to \infty$, it is optimal to select the largest feasible k. For an arbitrary A, the overall optimal solution and the corresponding estimated utility of the client can be found by taking the maximum over the possible values of k of $\sum_{i=1}^{k} w_i q_i^k - A \sum_{i=k+1}^{N} w_i$, where q_i^k denotes the quality allocation for stream i when downloading from k streams, as given by (11). Since there are at most $N - \lceil \frac{C}{Q_{max}} \rceil + 1$ possible values of k, and k' is nondecreasing in k, this problem can be solved in $O(N)$ time.

Multiples of Q_{min}: Suppose now that for all j, $q_{i,j} = jQ_{min}$, with $q_{i,n} = Q_{max}$. The solution approach for the case of infinitesimal granularity can be applied in this case as well, yielding a solution in $O(N)$ time, by replacing C in (11) by $\lfloor \frac{C}{Q_{min}} \rfloor Q_{min}$. Note that $C - \lfloor \frac{C}{Q_{min}} \rfloor Q_{min}$ cannot be allocated owing to the granularity of the available qualities.

D. Number of Candidate Solutions

The stall penalty A captures the extent to which a user prioritizes high average playback quality, versus avoiding playback stalls when switching streams. While the problem of how to determine an appropriate value for this parameter in some particular context of interest is outside the scope of this paper, we note that the stall penalty could be personalized, based on preferences observed for (or selected by) each user. In such a scenario, rather than just a solution to the optimization problem for one particular value of A, we would like to find a minimal set of solutions and the range of A values over which each is optimal, such that for any stall penalty A there is a solution in the set that is optimal for that A. We call such solutions "candidate solutions". In this sub-section we prove a result upper bounding the number of candidate solutions in a minimal set, for the baseline scenario. In the subsequent sub-section we address the problem of finding a minimal set and the optimality range for each set member.

Although the special cases considered previously provide some insights regarding the characteristics of candidate solutions, the problem of finding a set of candidate solutions is in general computationally expensive to solve. For example, even in the case where all quality levels are divisible by Q_{min} (but the quality levels do not include all multiples of Q_{min} up to Q_{max} as was assumed in the special case at the end of Section III-C) and $A = 0$ it is not always optimal to greedily assign qualities (and bandwidth) to the most heavily weighted streams. Furthermore, given a solution in which k streams are being downloaded from, it is not always optimal to use greedy re-allocation of bandwidth to obtain a solution (suitable for larger A) in which $k + 1$ streams are being downloaded from. Instead, the allocation for k streams may need to be revisited prior to such a bandwidth re-allocation.

For example, consider a scenario in which there are video quality levels $\mathcal{Q} = \{7, 6, 4, 1\}$ and stream weights (0.5, 0.251, 0.15, 0.1, 0.05, ...), and a solution in which the top two most heavily weighted streams are being downloaded from, with respective qualities (7, 6). A greedy approach to constructing, from this solution, a solution (for larger A) in which the top three streams are being downloaded from, with no greater total bandwidth usage, would compare the utility change that would result from moving to (6, 6, 1) versus that from moving to (7, 4, 1), and would select (6, 6, 1) since $(7 - 6) \times 0.5 < (6-4) \times 0.251$. However, this choice would need to be revisited when constructing a solution in which the top four streams are being downloaded from, since the optimal choice in this case is (7, 4, 1, 1). Clearly, it is not always optimal to greedily move bandwidth from high weight streams so as to enable downloading from additional (lower weight) streams.

There are, however, insights from the special cases considered previously that generalize. First, the number of streams being downloaded from in an optimal solution is monotonically increasing with the stall penalty A. Also, as established by Lemma 2 below, the number of candidate solutions in a minimal set can be upper bounded using C, Q_{min}, and Q_{max}. In the proof of Lemma 2 it is shown that there is at most one solution in any minimal set of candidate solutions, for any particular number of streams being downloaded from.

Lemma 2: Considering the full range of stall penalties A ($0 \leq A < \infty$), there are at most $k_{max} - k_{min} + 1$ candidate solutions in a minimal set, where $k_{min} = \lfloor \frac{C}{Q_{max}} \rfloor + \min[1, \lfloor \frac{C - Q_{max} \lfloor \frac{C}{Q_{max}} \rfloor}{Q_{min}} \rfloor]$ and $k_{max} = \min[N, \lfloor \frac{C}{Q_{min}} \rfloor]$.

Proof: (Lemma 2) Let k be the number of streams being downloaded from (i.e., $k = |\{i | f_i = 1\}|$). First, note that there is never any benefit from downloading from less than k_{min} streams. Any solution with $k < k_{min}$ is using no more than $kQ_{max} \leq (k_{min} - 1)Q_{max}$ bandwidth, which by definition of k_{min} is at most $C - Q_{min}$, allowing the solution to be improved by downloading from another stream with Q_{min} (or more). Second, it is not feasible to download from more than k_{max} streams (with Q_{min} or higher quality). Third, for a fixed k in the considered range there exists a quality allocation that is optimal for all A. To see this, note that for fixed k and any A, there is an optimal solution in which the first k streams (ordered by weight) are the ones being downloaded from (see proof of Lemma 1). The objective function (6) can now be split into two sums (i) $\sum_{i=1}^{k} w_i q_i$, and (ii) $-\sum_{i=k+1}^{N} w_i A$. Since the second sum is independent of the quality choices for the streams being downloaded from, the objective function for a fixed k is maximized by the set of q_i values that maximize the simplified objective function $\sum_{i=1}^{k} w_i q_i$, conditioned on $\sum_{i=1}^{k} q_i \leq C$. For k in the considered range there exists a solution to this problem, and it is obviously independent of A. This completes the proof. ■

Although $k = k_{max}$ is always optimal when $A \to \infty$, it is not necessarily the case that every one of the $k_{max} - k_{min} + 1$ values of k has some A value for which it is optimal. For example, consider the case of $C = 12$, twelve or more streams, and two qualities 10 and 1. In this case, $k_{min} = 2$ and $k_{max} = 12$, but only $k = 3$ (in which case the optimal qualities are 10, 1, and 1) and $k = 12$ (12 streams each with quality 1) are optimal for some A. Note in particular that the optimal k when $A = 0$ need not be k_{min}, as there could be additional bandwidth that could be allocated, after making the maximum possible allocations to the highest-weight k_{min} streams. In general, the size of a minimal set of candidate solutions can be substantially smaller than the bound of Lemma 2.

E. Finding Optimal Solutions

The proof of Lemma 2 suggests the following approach to finding a minimal set of solutions and the range of A values over which each is optimal: First solve the simplified optimization problem with objective function $\sum_{i=1}^{k} w_i q_i$, conditioned on $\sum_{i=1}^{k} q_i \leq C$, for all $k_{min} \leq k \leq k_{max}$, and then find the range of A values (if any) for which each solution is optimal.

Consider the two solutions for when k_1 and k_2 streams are being downloaded from, for some k_1, k_2 such that $k_{min} \leq k_1 < k_2 \leq k_{max}$. Since both solutions are feasible for any A, and the objective function (6) is a linear function

```
    INPUT: Capacity $C$, weights $w_i$, and utility function
    OUTPUT: Quality $q_i$ allocation
1.   $q_i \leftarrow 0, \forall i$
2.   while $(\sum_i q_i + \min_i \Delta_i^q) \leq C$
2.1.     $j \leftarrow \arg\max_i [\frac{w_i \Delta_i^u}{\Delta_i^q} | \sum_i q_i + \Delta_i^q \leq C]$
2.2.     $q_j \leftarrow q_j + \Delta_j^q$
3.   end while
```

Fig. 4. Pseudo-code of simple greedy algorithm.

of A, we can find the crossover point A_{k_1,k_2} where the k_2 solution becomes better than the k_1 solution by setting the two objective functions equal and solving for A. Denoting the quality allocation for stream i in the solution for when k streams are being downloaded from by q_i^k, this yields:

$$A_{k_1,k_2} = \frac{\sum_{i=1}^{k_1} w_i q_i^{k_1} - \sum_{i=1}^{k_2} w_i q_i^{k_2}}{\sum_{i=k_1+1}^{k_2} w_i}. \quad (12)$$

Although typically $0 < A_{k_1,k_2} \leq A_{k_2,k_3}$ for $k_1 < k_2 < k_3$, there are exceptions. For example, when a given k_1 is not optimal for any A, then A_{k_1,k_2} may be negative. For this reason, we calculate one transition point at a time.

The algorithm begins with $k = k_{min}$ and $A_k = -\infty$. The transition point $A_{k,k'}$ is found, such that $k' = \text{argmin}_{k<i\leq k_{max}}(A_{k,i}|A_k \leq A_{k,i})$. Since $A_k \leq A_{k,k'}$, the solution q_i^k is optimal for the interval $[\max[0, A_k], A_{k,k'}]$ (which could be null; i.e., if $A_{k,k'} < 0$). Then we set $A_k = A_{k,k'}$ and $k = k'$, and the above step is repeated to find the next transition point. The algorithm terminates when $k = k_{max}$, with the solution $q_i^{k_{max}}$ being optimal for all nonnegative A values greater than or equal to the last transition point.

By always finding the next transition point, this algorithm follows the convex curve of the optimal objective function (as a function of A) and so is guaranteed to find all optimal transition points. The algorithm must calculate at most $(k_{max} - k)$ candidate transition points in each iteration and there are at most $k_{max} - k_{min} + 1$ values of k that could be optimal for some A. The total number of transition point calculations is therefore upper bounded by $\frac{(k_{max}-k_{min})(k_{max}-k_{min}+1)}{2}$.

The number of candidate allocations that may need to be considered when finding the solutions for the simplified optimization problem (i.e., the q_i^k values for each k) is upper bounded by $\sum_{k=k_{min}}^{k_{max}} \binom{k+n-1}{n-1}$. This sum is equal to the number of monotonically non-decreasing paths in the $(n-1) \times k$ grids associated with the $(k_{max} - k_{min} + 1)$ LPs that need to be solved for $k_{min} \leq k \leq k_{max}$. We note, however, that it often may be faster to find the $(k_{max} - k_{min} + 1)$ solutions by solving the corresponding LPs using standard solvers, as typically most of the above candidate allocations can be pruned.

F. Greedy Heuristic

Motivated by the high computational complexity of the optimal solution, we next describe a simple greedy heuristic.

First, consider the general problem of maximizing the overall utility, given some available bandwidth C, and assuming a non-convex utility function. The greedy algorithm outlined in Figure 4 adds bandwidth to the stream i that maximizes $\frac{w_i \Delta_i^u}{\Delta_i^q}$, where Δ_i^u is the change in utility for stream i (in our baseline scenario equal to $q_i^{new} - q_i^{old}$ or $q_i^{new} + A$, depending on if $q_i^{old} > 0$ or not) and $\Delta_i^q = q_i^{new} - q_i^{old}$ is the change in quality of improving the quality of stream i one quality level. In each step, this choice greedily maximizes the increase in weighted utility per unit bandwidth. These greedy choices are then repeated as long as bandwidth can be allocated without exceeding the capacity constraint.

For the baseline scenario, this algorithm results in a solution that has very similar characteristics as the allocation illustrated in Figure 2. In fact, when the qualities are perfectly divisible (as in Section III-C) the greedy policy finds the optimal solution. To see this note that for our baseline utility function, the $\frac{w_i \Delta_i^u}{\Delta_i^q}$ is equal to w_i whenever a stream already has non-zero quality assigned. There will always be some number of streams with non-zero quality (say k), and these will be greedily loaded starting from one with highest weight.

Assuming again the baseline scenario, note that the above algorithm assumes that a solution is needed for just one particular value of A. For this case, a naive implementation of the algorithm (that uses $O(N)$ time for each evaluation of the $\min_i$ and $\arg\max_i$ terms and always starts from $q_i = 0$ for all i, for example) achieves a worst-case complexity of $O(nN^2)$. Optimizations to the time complexity are of course possible. To find a set of solutions that includes the greedy solution for any A value, we can find all candidate greedy solutions by simply considering all solutions in which exactly k streams first are allocated Q_{min} data and the remaining capacity $(C - kQ_{min})$ is greedily allocated among this subset of streams, as per the above algorithm. The same $O((k_{max} - k_{min})^2)$ algorithm as described for the optimal solution (Section III-E) can then be used to determine all the solution transition points.

IV. Numerical Characterization

This section presents a numerical characterization of the optimal solution under steady-state conditions. We assume that each video stream is available in four encoding qualities (250Kb/s, 500Kb/s, 850Kb/s, and 1300Kb/s) and the utility function normalizes all qualities relative to the lowest quality encoding Q_{min} = 250Kb/s. With these normalized units a client downloading all streams at rate Q_{min} has the normalized utility of 1 and the maximum possible normalized utility is 5.2 (when all streams are downloaded at the maximum quality).

Figure 5(a) shows examples of the optimal solution, for four different stall penalties. We have used C = 2000Kb/s, $N = 6$, and Zipf distributed weights with shape parameter $\alpha = 1$.[2] For each of the prefetching streams (indexed from 1-6) the y-axis shows the chunk quality determined for that stream. The Zipf model is used as an example distribution and is motivated by the popularity distributions observed in many diverse systems, including various content delivery applications and for the

[2]In practice, similar to other related systems, the switching probabilities can be estimated by monitoring the choices made by other clients and can be biased by recommendations and the user interface used for the switching, for example. We note that the problem of modeling switching probabilities and setting the best possible weights is orthogonal to this work, and we are not aware of any work modeling these probabilities in the stream bundle context. The Zipf model is only used as an example. Our design allows any technique (good or bad) to be used for setting weights. Section V-G evaluates and discusses the impact of the prediction accuracy.

channel switching in IPTV systems [14], [15]. Note that when $A = 0$ all of the bandwidth is used for the top two streams. As A increases there is downloading from more streams, and when A reaches 1.2, from all six streams. It turns out that the four candidate solutions shown here are the only candidate solutions, each optimal for a separate region of stall penalties.

Using the algorithm in Section III-E, the number of candidate solutions in a minimal set can be found for different numbers of streams N and Zipf parameters α. Example results are shown in Figure 5(b). Note that when $\alpha = 0$ all streams are given the same weight and there is only a single candidate solution, with $k = k_{max}$. The number of candidate solutions also reduces with more concave utility functions. For example, Figure 6 shows results with the normalized utility function $u(q_i) = (\frac{q_i}{Q_{min}})^{1/2} - (1 - f_i)A$, obtained using a MILP solver.

We next take a closer look at the stall penalty A and its impact on the optimal tradeoff between the weighted playback quality ($\sum_i w_i f_i q_i$) and the stall probability ($\sum_i w_i(1 - f_i)$). Figure 7 shows each of these two quantities and Figure 8(a) shows the normalized utility (where the playback quality is divided by Q_{min}) for the corresponding cases. As expected, both quantities decrease with increasing A according to a step function (each step corresponds to a new candidate solution), while the normalized utility follows a smoother curve.

Similar observations can be made when varying other parameters. Figures 8(b)-(d) show example results illustrating the impact of varying the number of streams N (in Figure 8(b)), the available bandwidth C (in Figure 8(c)), and the shape parameter γ of the normalized utility function $u(q_i) = (\frac{q_i}{Q_{min}})^{\gamma} - (1 - f_i)A$ (in Figure 8(d)). Note that the normalized utility function shows a diminishing decrease with increasing N, diminishing increase with increasing C, and that the overall utility differences between optimal solutions decrease with a more concave utility function (smaller γ).

We also have evaluated the above test cases with the greedy heuristic. While it is easy to show (by construction of counterexamples) that the greedy algorithm is not optimal, the greedy algorithm did actually find the optimal solution in all cases considered here. This suggests that it may be a good approximation algorithm for our purposes.

V. Experimental Implementation and Evaluation

We have implemented a proof-of-concept system[3] based on the open source OSMF framework[4].

A. System Design

At a high level, our implementation (i) feeds the currently played stream straight into the player buffer, (ii) prefetches the alternative streams into the cache (from which they can quickly be fetched when switching streams), (iii) keeps track of the workahead of the currently played stream, the prefetched workahead of the alternative streams, as well as the estimated download rate, and (iv) when a client switches streams, flushes the playback buffer, and loads the initial chunks of the new stream from the cache.

Our modified OSMF player uses both standard libraries and a custom built class that manages most aspects of our implementation. First, at the end of every chunk download, the client estimates the available bandwidth C_{est} using an exponentially weighted moving average (EWMA) with $\alpha = 0.4$.[5] For simplicity, the client downloads chunks back-to-back over a single TCP connection. As outlined in Section II-C, we first calculate the dynamic bandwidth shares C_{play} and C_{pref} (Section II-C1), then the individual q_i values (Section III), before finally using our round-robin technique (Section II-C2) to select the next chunk (and quality) to download next. To ensure fast calculations, we use the greedy algorithm (Section III-F) for the q_i calculations in our implementation.

The player continually tracks the playback point of the current stream. When a stream switch is initiated, the current time is passed to the new player instance, which automatically seeks to the current play point. This ensures that the stream switch moves to a different video (camera/view) while maintaining time synchronization among streams.

The client uses an intelligent buffer management solution to ensure that transitions are as seamless as possible. Every newly played stream requires a new player instance. Instead of waiting for a new stream to commence playback, the client continues playing the video on the older instance. Once the buffer of the new instance is sufficiently full, it is brought to the foreground and the older instance is deleted, thereby masking a significant portion of the stream switching delay.

B. Experimental Setup

We use a Firefox browser (v 26.0) configured with the browser cache in RAM. We use Flash Media Server (v 4.5) to host our video and Dummynet [19] to control the total bandwidth capacity and round-trip time (RTT) to the server. For the majority of the presented results, and unless otherwise stated, we use 6000Kbps and 50ms RTT. The client implementation runs on Windows 7 and the server is hosted on a PC running Ubuntu 14.10.

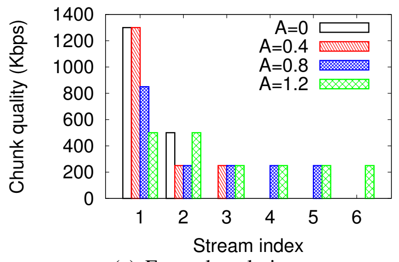
For each stream, we use parts of the Big Buck Bunny video encoded according to Adobe's media encoding recommendations and packaged into chunks of 4 second durations, but with a different name and individual manifests for each copy. Each of these videos is encoded at four bit rates (250Kbps, 500Kbps, 850Kbps, and 1300Kbps).

In our default scenario, streams are weighted according to a Zipf distribution (with $\alpha = 1$), with the streams ordered by their relative distance from the currently played stream as measured by the difference between the stream indices (modulo N). Such an ordering could correspond to a scenario in which the stream indexing reflects similarity in viewpoint, and users usually make incremental changes in viewpoint.

[3]Our source code and system framework are available at http://www.ida.liu.se/~nikca89/papers/tmm17.html.

[4]http://sourceforge.net/projects/osmf.adobe/

[5]The choice of using a simple EWMA allows easy head-to-head policy comparison. However, we note that any rate estimator potentially could be used here and that more advanced rate and throughput estimation techniques [16], [17], [18] would allow more accurate estimation of C_{est}; hence, further improving the performance of all the policies.


(a) Example solutions

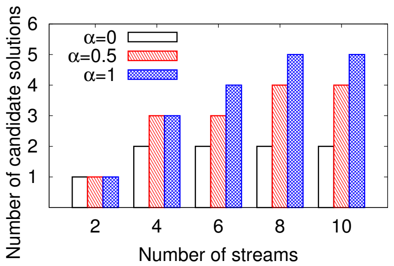

(b) Number of candidate solutions

Fig. 5. Example solutions for different stall penalties A (when $\alpha = 1$) and number of candidate solutions for different weight skews α and streams N.

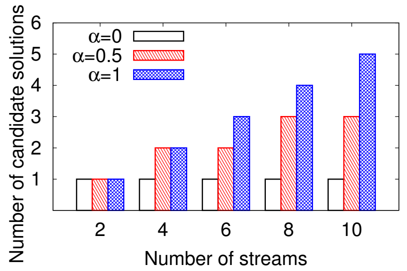
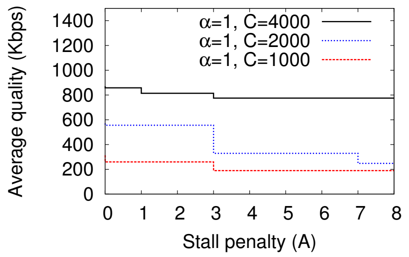


Fig. 6. Number of candidate solutions when using a square-root-based utility function, for different weight skews α and streams N.


(a) Weighted playback quality

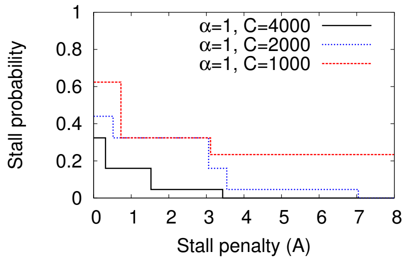
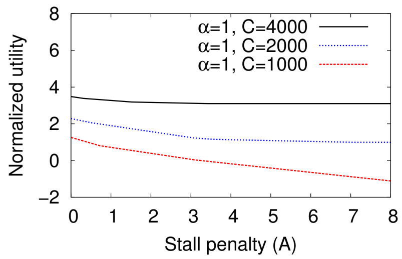
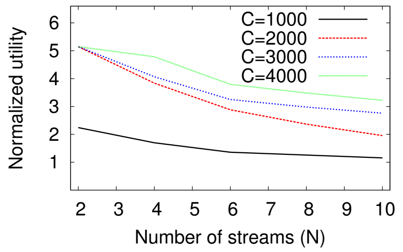
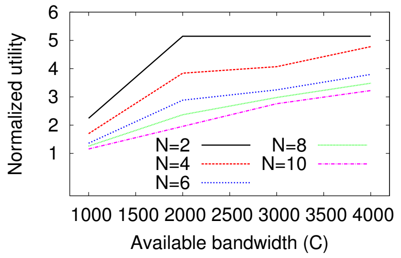

(b) Stall probability

Fig. 7. Breakdown into weighted playback quality and stall probability.


(a) Stall penalty


(b) Number of streams


(c) Available bandwidth capacity

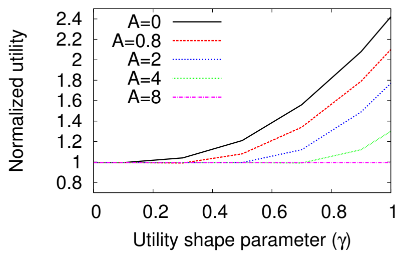

(d) Utility function

Fig. 8. Impact of parameters on the normalized utility.

We also perform experiments with scenarios in which all streams are equally likely to be selected and scenarios in which the selection bias is even greater than with the Zipf distribution. For the first case, we use uniform weights. This case also helps in understanding the case when it is not possible to predict the switching probabilities. For the other extreme, we use a geometric distribution. In particular, each stream is given half the weight of an (adjacent) stream one index closer to the currently played stream, with relative proximity being measured by the difference in stream index modulo N. With this choice, the weight of a stream that is k streams away from the currently played stream has a weight proportional to $\frac{1}{2^k}$. In general, we expect the bias to be between these two extremes.

Policies: To put the dynamics of our adaptive prefetching technique in perspective, in addition to our prefetching policy framework (called "Adaptive"), we also include results for a "Vanilla" player that uses the default settings of a regular OSMF player, and hence goes idle (creating "off" periods) when reaching T_{max}, as well as for a baseline policy (RR-OFF) that simply downloads alternative streams in round-robin order when the currently played stream is in an off period. Here, an "off period" is the time during which the Vanilla player would drain its buffer from T_{max} back to T_{min}, where T_{min} and T_{max} are the main parameters used in the OSMF framework, and the quality of each chunk prefetched by the RR-OFF player is selected based on the full download rate estimates and using the same quality selection rule as used for the currently played stream. This policy is based on our previous work [20] that shows how downloading chunks of other videos during off periods, as with the RR-OFF policy, can greatly reduce the startup times of alternative videos in the context of video-on-demand. For the Adaptive and RR-OFF implementations we use $T_{min} = 4$ seconds and $T_{max} = 30$ seconds, whereas for the Vanilla player we use the default OSMF parameters $T_{min} = 4$ seconds and $T_{max} = 6$ seconds.

C. Validation of Stream Switching

To understand the impact of stream switching and how seamless this can be made, we instrumented the player to collect low-level time measurements and analyzed a large number of stream switching events. Considering our default scenario, the average time observed from when the client selects to play the new stream until the new stream commences playback is 0.93 seconds (SD=0.21). However, during the majority of this time period, the previous stream would have still been playing, allowing the transition interval to be masked.

In fact, the average time taken to change player instances and resume playback was only 0.054 seconds (SD=0.016). This is the effective time for which there is no video playback.

Our prefetching strategies play an important part in keeping the transition time small by ensuring that the landing chunk is available in the cache. With a policy that does not prefetch, the load time will be a function of the available bandwidth. In our experiments without prefetching at 2Mbps and 50ms RTT the average load time was 1.98 seconds (SD=0.45), making the switch delay much more apparent to the user even though the stall itself is similar (0.056 ± 0.010 seconds).

D. Buffer Occupancy under Example Scenarios

We next present some measurement results illustrating the operation and performance of our system for the initial (transient) phase of two example scenarios. Figure 9 shows the buffer occupancy for the currently played stream (0) and six alternative streams (1-6) as a function of time for the initial phase of a scenario without switching. Results are shown for both the RR-OFF (Figure 9(a)) and our Adaptive (Figure 9(b)) policy. We use a penalty $A = 1.6$ for the Adaptive policy. The most important observation here is that with the Adaptive policy the most likely alternative streams are prefetched roughly twice as quickly as with the RR-OFF policy. This difference is in part due to the RR-OFF policy not beginning to prefetch alternative streams until reaching T_{max}. In addition, since the RR-OFF policy does not take into account each stream's rightful share, but instead uses the full download rate estimates to select chunk qualities, it will typically prefetch each stream at a higher quality, reducing the amount of buffer that can be built up for each stream. Again, as noted by Huang et al. [11], for example, a large buffer (reservoir) is important to avoid unnecessary stalls.

Figure 9(c) shows example results for the Adaptive policy for a scenario where the client switches from stream 0 to stream 1 at the 30 second mark, and then to stream 2 at the 60 second mark. Note that the buffer conditions adapt well to changes in which stream is being currently played, and that there is substantial prioritization of the closest (according to our ordering) streams. This is perhaps made even clearer by observing how the quality of the buffered chunks changes with time (Figure 10). First the quality of the buffered stream 0 chunks is highest, then the quality of the stream 1 chunks, and finally the stream 2 chunks have the highest quality. Figure 11 shows the buffer occupancies of the differently ranked streams as CDFs. The observed similarities in CDFs imply that there is almost as much buffered content for the highest priority alternative streams as for the played stream, as desired. These results are encouraging and show that our system works well.

E. Longer Duration Experiments

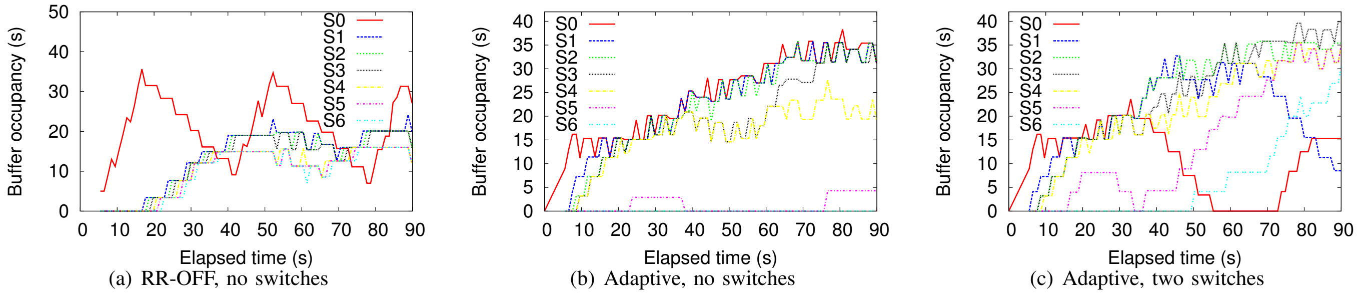
We next present some results from longer duration experiments with more stream switches, considering both general behavioral differences among the policies, and the impact of the bias in the stream switching probability distribution. In these experiments the client switches streams with probability 0.5 every 30 seconds, over a 360 second (6 minute) experiment duration. Following each switch, over the next 30 seconds we track the playback quality, buffer occupancy associated with the currently played stream, and the probability that a stall would occur if the client immediately switched streams again. Figure 12 shows the results averaged over ten runs with each of several stream switching bias and policy combinations.

Referring to Figure 12(b), we note that our Adaptive policy consistently maintains a larger buffer for the played stream than the Vanilla player and the RR-OFF policy, for all the switching probability models. The larger buffer is important for protecting against stalls caused by variations in bandwidth, variations in encoding (as in the case of VBR, for example), or other unforeseen connection variations and interruptions.

Perhaps most importantly, note that the Adaptive policy increases both the buffer size (Figure 12(b)) and playback quality of the played video (Figure 12(a)), while simultaneously maintaining a relatively low stall probability (Figure 12(c)). For example, in these results, the stall probability is almost always at least twice as high for RR-OFF than for Adaptive, and often significantly greater. Comparing the results for a Zipf selection bias when $A = 1.6$ (our default) versus $A = 3.2$, note that the stall probability can be further reduced with our policy, simply by giving more weight to the importance of avoiding stalls. The higher protection against stalls is also evident from the buffer occupancy for the new played stream immediately after a switch, as shown by the values for a time of 0 in Figure 12(b). This buffer size corresponds to the average (weighted) buffer occupancies of the alternative streams, immediately prior to the switch. Again, the Adaptive policy consistently has higher initial buffer values.

The lowest stall probabilities (Figure 12(c)) are observed for the Adaptive policy with uniform random stream switching probabilities. In this case, the available bandwidth for prefetching is spread more uniformly among the alternative streams, rather than prefetching higher quality chunks from only a few streams as occurs with highly skewed probabilities. This is reflected in the lower playback quality (Figure 12(a)) observed just after a switch in the case of uniform random probabilities, compared to with Zipf or geometric probabilities. In general, for all of the switching probability distributions, the playback quality of the played stream (Figure 12(a)) increases as an approximately concave function, reaching the qualities observed with the RR-OFF policy within the first 30 seconds after a switch. The difference in shape between the curves for the Vanilla player and the RR-OFF policy primarily comes from the Vanilla player (i) not having any prefetched chunks, (ii) interpreting a switch as a download of a new video (followed by a seek to the current playpoint), and (iii) therefore starting the initial downloads at a low quality, whereas the other policies almost always have some prefetched chunks.

In contrast to the Adaptive policy, the RR-OFF policy almost exclusively downloads at the highest possible quality (as seen in Figure 12(a) for the played stream), not allowing as much prefetching of alternative streams. In addition to substantially higher stall probability at switching instances (Figure 12(c)), we note that the smaller average buffer size for the currently played stream (Figure 12(b)) makes the RR-OFF policy much less resilient to other variability and



(a) RR-OFF, no switches (b) Adaptive, no switches (c) Adaptive, two switches

Fig. 9. Buffer occupancy for example scenarios with RR-OFF and Adaptive policy.

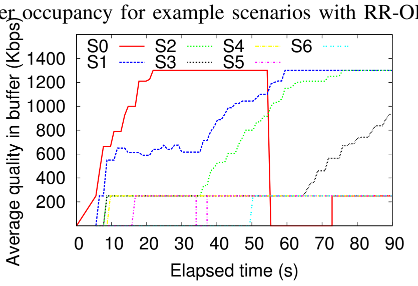


Fig. 10. Average chunk quality encoding in two-switch scenario with Adaptive policy.

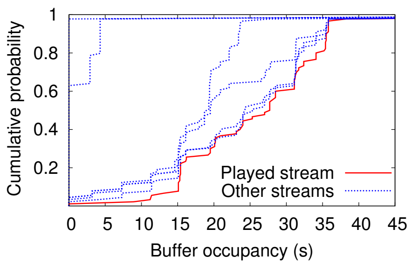
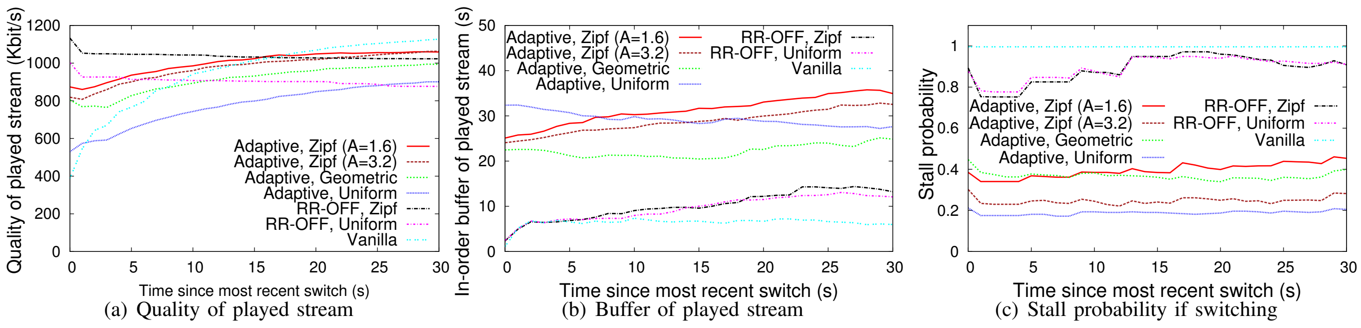


Fig. 11. Buffer occupancy distributions with Adaptive policy.



(a) Quality of played stream (b) Buffer of played stream (c) Stall probability if switching

Fig. 12. Average playback conditions as a function of the time since the most recent switch, for each of the different policies and switching probability biases.

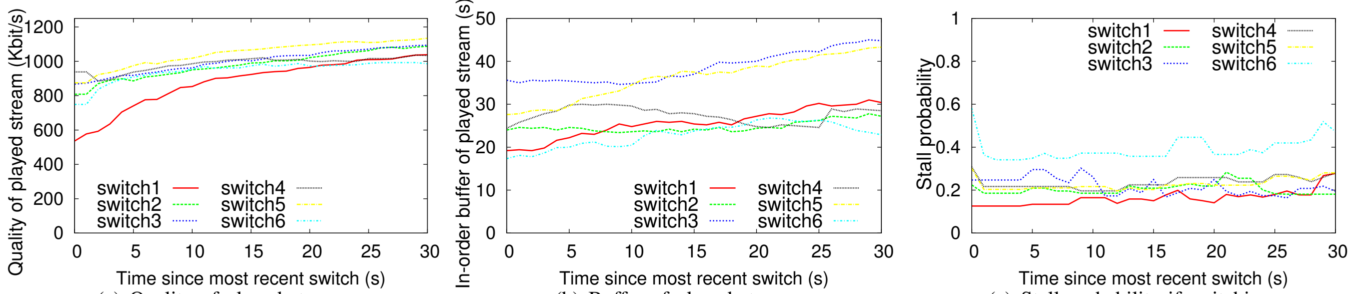
unforeseen bandwidth interruptions, for example. The RR-OFF policy usually downloads at the highest possible quality since it does not take into account the bandwidth needed for the downloading of chunks from other streams. Furthermore, the large difference in average buffer size of the played stream can be explained by differences in the general protocol dynamics. In particular, whereas the Adaptive policy tries to maintain a steady buffer of the played stream, the RR-OFF policy cycles between T_{min} and T_{max}, going back and forth between on-off periods. This cycling results in the average buffer size observed for the RR-OFF policy being roughly half that for the Adaptive policy. Given that stalls typically are the main deterrent to good quality of experience, the larger and steadier buffer occupancy with the Adaptive policy is therefore desirable.

While the stall probabilities and playback quality capture the first-order metrics of the performance, we note that the quality variations also can impact the perceived user experience. Referring to Figures 12(a) and 12(c), we note that Vanilla has worse quality variation and will always stall, while RR-OFF has more consistent quality, but at the cost of unacceptable stall probability. In contrast to these, our Adaptive policy tries to adapt the quality selection across all streams and over time so as to achieve a good tradeoff between quality and stall probabilities, resulting in intermediate quality variations over time (typically starting at a lower quality after a switch and then increasing the quality over time). In future work, we will look at enhancements of our proposed techniques, which more fully consider the issue of quality variations of the playback quality itself, given similar stall probabilities, for example.

F. *Longitudinal Characterization*

Naturally, it may take some time before the buffers are first filled, and the system reaches some form of steady state. To investigate this longitudinal aspect, we next break down the results from one of our longer duration experiments with multiple stream switches, using the Adaptive policy, according to how many switches had occurred prior to each 30 second tracking interval. Results are shown in Figure 13 for the first six switches, and for a Zipf stream selection probability distribution. Results with other biases are similar.

Although there are non-negligible quantitative differences, the qualitative behavior after each of the six switches is similar. In such experiments we have not observed any clear trends after the first switch or two, suggesting that with the Adaptive policy, the system may reach steady state relatively quickly. Most of the differences in the curves appear to be random in



(a) Quality of played stream (b) Buffer of played stream (c) Stall probability if switching

Fig. 13. Average playback conditions with the Adaptive policy as a function of the time since the most recent switch, after each of the first six switches.

nature, due to the random selection of an alternative stream when a stream switch occurs. This is, for example, the reason for the higher stall probabilities and lower buffer occupancies after the sixth stream switch in the results in Figure 13.

G. Impact of Prediction Accuracy

While the problem of modeling and estimating the switching probabilities is orthogonal to this work and our system solution is independent of how these probabilities are predicted, the prediction accuracy can impact the absolute performance of the system. We next take a closer look at this performance impact. First, referring back to Figure 12, we note that the performance of the Adaptive policy is relatively good even when the probabilities are completely unknown (as exemplified by the "uniform" curves). This suggests that assuming uniform probabilities (or applying a large penalty A) can be a good solution when probabilities are difficult to predict.

Second, in Table II we compare results for cases when the system has predicted the probabilities to be Zipf and uniform, respectively, and the true probabilities either are a match or they follow the other example distribution. For the cases in which the distributions match, the results correspond to the left-most and right-most values for the corresponding plots in Figure 12. We again observe that assuming more uniform probabilities (regardless if correct) results in smaller stall probabilities. A similar effect can be achieved with the Adaptive policy by using a larger penalty value A.

Overall, the Adaptive policy only sees minor degradations from incorrect prediction and consistently outperforms the RR-OFF policy for all metrics shown in Table II except the initial playback quality immediately after a switch. Again, the somewhat lower initial playback quality just after a switch (e.g., left-most points in Figure 12(a)) is due to more weight being given to avoid stalls. These results are encouraging, as they show that much of the performance improvements over RR-OFF can be achieved even when the system is not accurately predicting the magnitude of the biases. Note also that good prediction algorithms could be expected to make better predictions than the simple uniform assumption. Finally, we note that determining the best methods for estimating these probabilities is an interesting open problem for future work.

H. Bandwidth Variations

We conclude our evaluation with experiments in which we either use competing traffic, or synthetically vary the available bandwidth over time using Dummynet. Table III summarizes example results for Adaptive and RR-OFF. (Results for Vanilla are consistently much worse than both Adaptive and RR-OFF.) For the scenarios with competing traffic we set up a second server from which the client downloads one or more large files in parallel using TCP. We use the same default RTT (50ms) for both servers as in the default scenario, but scale the bandwidth of the shared bottleneck link proportional to the number of parallel connections. For the low-variability Dummynet scenario we use a trace in which we randomly toggle between one of three bandwidths every 30 seconds, where the set of bandwidths are 3,000, 6,000, and 9,000 Kbit/s. For the high-variability scenario we toggle to a new bandwidth every 6 seconds, each time randomly picking from the set 2,000, 4,000, 6,000, 8,000, and 10,000 Kbit/s. The high-frequency scenario uses the same bandwidth-hopping sequence as the later scenario, but changes occur at twice the rate.

We have also evaluated the different policies under real-world commuter traces, collected in 2011 while using different modes of transport in Oslo [21]. So as to make the average available bandwidth more representative of likely current scenarios, as well as more similar to those in our other tests, bandwidths were inflated. Bandwidths for the bus trace were inflated by a factor 3, for the car trace by a factor 2.5 (equal to the average bandwidth increase in mobile networks between 2013 and 2016 as per the Swedish speed test service Bredbandskollen [22]), and for the metro trace by a factor 6.2 (equal to the average bandwidth increase between 2011 and 2016). After these inflations, the bus trace has an average bandwidth of 6,209 Kbit/s and a standard deviation of 2,057 Kbit/s. The corresponding values for the inflated car trace are 5,250 Kbit/s and 1,740 Kbit/s, respectively, and for the metro trace they are 6,062 Kbit/s and 2,541 Kbit/s, respectively. When interpreting the results and comparing the impact of bandwidth variations, it is important to remember that the average available bandwidth will differ for the competing traffic cases compared to the other cases, and the real-world traces do not have exactly the same average available bandwidth.

As expected, high variability negatively impacts performance. However, we again observe that the Adaptive policy consistently outperforms the RR-OFF policy. In summary, our results show that our system provides a good step in the right direction for providing seamless stream bundling services.

TABLE II. IMPACT OF PREDICTION ERRORS

	Playback quality (Kbit/s)				Buffer played stream (s)				Stall probability			
	Right after switch		30s after switch		Right after switch		30s after switch		Right after switch		30s after switch	
Assumed/true bias	Adaptive	RR-OFF	Adaptive	RR-OFF	Adaptive	RR-OFF	Adaptive	RR-OFF	Adaptive	RR-OFF	Adaptive	RR-OFF
Zipf/Zipf (correct)	874	889	1,059	794	25.0	2.33	34.9	13.2	0.38	0.89	0.45	0.90
Zipf/Uniform (wrong)	764	775	1,022	684	28.9	2.44	29.1	12.0	0.33	0.87	0.59	0.90
Uniform/Uniform (correct)	530	775	901	684	32.3	2.44	27.6	12.0	0.21	0.87	0.20	0.90
Uniform/Zipf (wrong)	628	889	911	794	17.8	2.33	21.9	13.2	0.21	0.89	0.29	0.90

TABLE III. EXAMPLE CASES WITH DIFFERENT BANDWIDTH VARIABILITY

	Playback quality (Kbit/s)				Buffer played stream (s)				Stall probability			
	Right after switch		30s after switch		Right after switch		30s after switch		Right after switch		30s after switch	
Scenario	Adaptive	RR-OFF	Adaptive	RR-OFF	Adaptive	RR-OFF	Adaptive	RR-OFF	Adaptive	RR-OFF	Adaptive	RR-OFF
Default scenario	874	1,133	1,059	1,023	25.1	5.0	34.9	14.1	0.38	0.89	0.45	0.90
One (1) competing flow	828	713	1081	913	28.9	6.5	38.8	9.9	0.16	0.46	0.32	0.96
Two (2) competing flows	775	886	1019	1013	26.1	7.7	27.6	15.2	0.25	0.37	0.44	0.94
Three (3) competing flows	907	769	1063	994	26.3	9.0	28.8	13.9	0.15	0.52	0.28	0.91
Low-variability emulation	827	977	999	774	19.8	3.3	23.3	7.9	0.27	0.58	0.49	0.94
High-variability emulation	865	950	1,033	835	15.3	5.0	20.5	8.6	0.29	0.55	0.61	0.97
High-frequency emulation	786	598	1,000	990	14.9	7.7	19.4	12.8	0.32	0.5	0.64	0.96
Bus trace (x3) emulation	829	647	1,036	970	20.5	14.25	24.8	22.0	0.27	0.50	0.43	0.83
Car trace (x2.5) emulation	777	427	988	638	13.7	3.4	20.0	6.8	0.45	0.65	0.67	0.97
Metro trace (x6.2) emulation	848	457	1,025	818	21.1	4.3	23.7	11.0	0.42	0.53	0.58	0.97

VI. RELATED WORK

While the concept of HAS-based multi-video stream bundles is new, multi-view streaming is not. Most previous work concerning multi-view streaming has focused on various video coding aspects, including how to best encode multiple viewpoints into a single video stream [1], [2], coding schemes that facilitate quick viewpoint switching [23], [24], [25], and that combine multiple texture/color and depth map streams to achieve free viewpoint streaming, including of virtual viewpoints [26], [5], [27], [28], [29]. In the context of 3D-TV, both multi-layered solutions [30] and prefetching information of alternative viewpoints [31] have been shown useful.

Others have considered coding or network coding solutions for reduced bandwidth usage in multi-user scenarios [32], [33]. Both multicast-based delivery protocols [34] and peer-assisted solutions [35], [36] have also been proposed, typically delivering different views of a fixed encoding quality per stream over parallel overlay networks, but also solutions that implement pan/tilt/zoom functionality into the delivery protocol, taking into account the region of interest [37].

In contrast to the above works, we present a client-driven HAS-based multi-video streaming approach and solution in which the user selects between a finite number of viewpoints (one per stream in the stream bundle), and in which each stream easily can be individually recorded and encoded. The use of HAS allows us to optimize the bandwidth allocated to both the played video stream and the prefetched alternative video streams (alternative view points) such as to provide highest possible playback quality and stall-free switching.

In the context of linear video (without viewpoint selection), much work has been done on HAS rate adaptation [38], [39], [16]. For example, researchers have proposed and evaluated various rate estimation techniques [17], [18], [11] beyond the basic EWMA-based techniques used here. Others have demonstrated the value of good rate estimations [40], and shown that playback stalls and frequent quality switches are the most common reasons for low user satisfaction and video abandonment during playback [41], [42]. These latter insights highlight the importance of careful prefetching in our stream bundle context. Other orthogonal techniques to reduce the stall times, proposed and evaluated for linear video streaming of a single video, rather than for stream bundles, include data-driven throughput prediction techniques [16] and techniques that combine buffer and rate estimations [43] going beyond the reservoir approach [11] that inspired our design. Applying such additional optimization techniques in the context of our stream bundling system presents interesting future work, including how to best balance more complex objectives related to the played stream (focus of the above works) and the streams that the client may switch to (considered here).

HAS-based prefetching solutions have previously been applied to both branched video streaming [44] and to preload the beginning of alternative videos [20]. With branched video [44] (sometimes called nonlinear video [45], [46]) clients can interactively select their own path through a video in which the plot sequences can be described as a graph structure. A related concept is hypervideo, in which hyperlinks to other related video segments are presented during video streaming. Unlike with branched video, transitions do not need to be seamless, but prefetching is still important so as to minimize latency. Prior work has developed hypervideo prefetching policies using a Markov Decision Process framework [47], and has considered the use of immersive video in which the users can also change direction and zoom in/out so as to interact with the environment [48]. Other related work has used a modified player to combine video snippets based on tagged videos and user search queries into a single personalized video stream [49]. In contrast to these works, with stream bundles, as considered here, the clients are allowed to switch between the different streams at any arbitrary point in time and the new playback point must be aligned with that in the prior stream (not the start of the video or branch of the video). This system design also presents a new and challenging prefetching problem, for which we present optimized solutions.

Other related works have used HAS for multi-view streaming [3], [4], region-of-interest (ROI) based tiled streaming [9], [10], [50], [51], or systems in which video from multiple cameras is combined into personalized wide-field-of-view panorama video experiences [52]. All these works combine video streams from multiple cameras or use tiles from ultrahigh resolution omni-directional (360°) or panoramic video to provide personalized interactive streaming experiences. However, none of these works consider the prefetching problem associated with stream switching. In particular, while these solutions perform adaptation based on the available bandwidth, they do not perform adaptation of what video data is retrieved by the client based on the stream switching probabilities so as to balance the estimated playback quality and the probability of playback interruptions at the time of stream switching. Balancing this tradeoff is important for the user perceived quality of experience. The stream bundle concept provides a simple abstraction under which client-driven adaptation and quality rate optimization can be performed along both these dimensions. Interesting future work could include attempting to extend the optimization framework developed here to the context where clients may combine streams from multiple parallel streams (or tiles) to make up the current viewpoint.

Perhaps most closely related to our work is the work of Zhang et al. [53], and that of Hamza and Hefeeda [6]. Zhang et al. [53] combine HAS and multi-view streaming, but for a context in which clients view all video streams concurrently, rather than switch from stream to stream as they change viewpoints. In the work of Zhang et al., each stream is allocated a separate buffer. A unified scheduling heuristic is proposed, with complexity $O(N^3)$, where N is the number of video streams. The number of streams is expected to be small since only a small number of videos would typically be concurrently presented on the same screen, and the experiments presented are only for two video streams.

Hamza and Hefeeda [6] also employ HAS, but for free viewpoint video streaming. With free viewpoint video streaming, viewers are able to choose viewpoints that have not been captured by cameras, but instead must be synthesized from the reference views that have been captured. Hamza and Hefeeda focus on the problem of adaptively determining which reference views should be requested at each point in time, and for each of these the quality levels for the associated texture and depth streams, so as to best support view synthesis. Again, neither of these works consider the optimized prefetching problem of stream bundling systems, in which the playback quality of the played stream must be balanced against the probability of playback interruptions when switching streams.

VII. Conclusions

This paper presents a novel multi-video stream bundle framework for interactive video playback that allows users to dynamically switch among multiple parallel video streams capturing the same scenes from different viewpoints. Our solution includes an optimization framework and an adaptive protocol for chunk download scheduling that allows the quality of each parallel stream to be adapted based on the user's current bandwidth conditions and stream switching probabilities. The framework balances the estimated playback quality and the probability of playback interruptions at the time of stream switching. Through analytic, numerical, and experimental study, we show that our optimization framework and buffer management solution are able to achieve effective prefetching and close to seamless playback switching when there is sufficient bandwidth to prefetch the alternative streams.

In this paper we have assumed known weights and utility functions for each client. Interesting future work includes how to dynamically assign personalized weights for the most likely streams to switch to next (e.g., by the design of data-driven methods for determining stream switching probabilities), and the design of (online) algorithms for determining appropriate values of the penalty parameter A. Other interesting future work could be to integrate social media and other aspects that may impact the switching probabilities [54].

Finally, we note that our technique is client-driven and does not depend on whether the service is provided in the cloud, by CDN servers, or a single origin server, for example. There is much complementary work concerning video delivery from the cloud [55] and/or with proxy support [56]. Future work could concern exploiting proxy support and/or cloud-computing capabilities in delivery of multi-video stream bundles.

Acknowledgements

This work was supported by funding from the Swedish Research Council (VR), the Center for Industrial Information Technology (CENIIT), and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

[1] N. Cheung, A. Ortega, and G. Cheung, "Distributed source coding techniques for interactive multiview video streaming," in *Proc. PCS*, 2009.

[2] G. Cheung, A. Ortega, and T. Sakamoto, "Coding structure optimization for interactive multiview streaming in virtual world observation," in *Proc. IEEE MMSP*, 2008.

[3] T. Su, A. Sobhani, A. Yassine, S. Shirmohammadi, and A. Javadtalab, "A DASH-based HEVC multi-view video streaming system," *Journal of Real-Time Image Processing*, vol. 12, no. 2, pp. 329–342, 2016.

[4] M. Zhao, X. Gong, J. Liang, J. Guo, W. Wang, X. Que, and S. Cheng, "A cloud-assisted DASH-based scalable interactive multiview video streaming framework," in *Proc. PCS*, 2015.

[5] X. Xiu, G. Cheung, and J. Liang, "Delay-cognizant interactive streaming of multiview video with free viewpoint synthesis," *IEEE Trans. on Multimedia*, vol. 14, no. 4, pp. 1109–1126, 2012.

[6] A. Hamza and M. Hefeeda, "Adaptive streaming of interactive free viewpoint videos to heterogeneous clients," in *Proc. ACM MMSys*, 2016.

[7] A. Mavlankar, P. Agrawal, D. Pang, S. Halawa, N.-M. Cheung, and B. Girod, "An interactive region-of-interest video streaming system for online lecture viewing," in *Proc. PV*, 2010.

[8] R. van Brandenburg, O. Niamut, M. Prins, and H. Stokking, "Spatial segmentation for immersive media delivery," in *Proc. IEEE ICIN*, 2011.

[9] O. A. Niamut, E. Thomas, L. D'Acunto, C. Concolato, F. Denoual, and S. Y. Lim, "MPEG DASH SRD: Spatial relationship description," in *Proc. ACM MMSys*, 2016.

[10] J. Devloo, N. Lamot, J. Van Campen, E. Weymaere, S. Latré, J. Famaey, R. Van Brandenburg, and F. De Turck, "Design and evaluation of tile selection algorithms for tiled HTTP adaptive streaming," in *Proc. IFIP AIMS*, 2013.

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

[12] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *Proc. ACM SIGCOMM*, 2015.

[13] J. Summers, T. Brecht, D. L. Eager, and A. Gutarin, "Characterizing the workload of a Netflix streaming video server," in *Proc. IEEE IISWC*, 2016.

[14] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatriain, "Watching television over an IP network," in *Proc. IMC*, 2008.

[15] T. Qiu, Z. Ge, S. Lee, J. Wang, Q. Zhao, and J. Xu, "Modeling channel popularity dynamics in a large IPTV system." in *Proc. ACM SIGMETRICS/Performance*, 2009.

[16] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proc. ACM SIGCOMM*, 2016.

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

[18] L. Zhi, Z. Xiaoqing, J. Gahm, P. Rong, H. Hao, A. Begen, and D. Oran, "Probe and adapt: Rate adaptation for HTTP video streaming at scale," *IEEE JSAC*, vol. 32, no. 4, pp. 719–733, 2014.

[19] L. Rizzo., "Dummynet: A simple approach to the evaluation of network protocols," *ACM CCR*, vol. 27, no. 1, pp. 31–41, 1997.

[20] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri, "Bandwidth-aware prefetching for proactive multi-video preloading and improved HAS performance," in *Proc. ACM Multimedia*, 2015.

[21] 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. MoVid*, 2012.

[22] P. Davidsson, "Bredbandskollen: Mobil surfhastighet i Sverige 2016 - Technical report," 2016.

[23] X. Guo, Y. Lu, W. Gao, and Q. Huang, "Viewpoint switching in multiview video streaming," in *Proc. IEEE ISCAS*, 2005.

[24] G. Cheung, A. Ortega, and C. Ngai-Man, "Generation of redundant frame structure for interactive multiview streaming," in *Proc. PV*, 2009.

[25] H. Huang, B. Zhang, S. Chan, G. Cheung, and P. Frossard, "Coding and replication co-design for interactive multiview video streaming," in *Proc. IEEE INFOCOM*, 2012.

[26] C. Fehn, "Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV," in *Proc. SD&A*, 2004.

[27] T. Maugey and P. Frossard, "Interactive multiview video system with low complexity 2D look around at decoder," *IEEE Trans. on Multimedia*, vol. 15, no. 5, pp. 1070—-1082, 2013.

[28] J. Chakareski, V. Velisavljevic, and V. Stankovic, "User-action-driven view and rate scalable multiview video coding," *IEEE Trans. Image Process*, vol. 22, no. 9, pp. 3473–3484, 2013.

[29] A. D. Abreu, L. Toni, N. Thomos, T. Maugey, F. Pereira, and P. Frossard, "Optimal layered representation for adaptive interactive multiview video streaming," *Journal of Visual Communication and Image Representation*, vol. 33, pp. 255–264, 2015.

[30] A. M. Tekalp, E. Kurutepe, and M. R. Civanlar, "3DTV over IP: End-to-end streaming of multiview video," *IEEE Signal Process. Mag.*, vol. 24, no. 6, pp. 77—-87, 2007.

[31] E. Kurutepe, M. Civanlar, and A. Tekalp, "Client-driven selective streaming of multiview video for interactive 3DTV," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 11, pp. 1558–1565, 2007.

[32] T. Fujihashi, Z. Pan, and T. Watanabe, "UMSM: A traffic reduction method on multi-view video streaming for multiple users," *IEEE Trans. Multimedia*, vol. 16, no. 1, pp. 228—241, 2014.

[33] L. Toni, N. Thomos, and P. Frossard, "Interactive free viewpoint video streaming using prioritized network coding," in *Proc. IEEE MMSP*, 2013.

[34] J. Chakareski, "Adaptive multiview video streaming: challenges and opportunities," *IEEE Commun. Mag.*, vol. 51, no. 5, pp. 94–100, 2013.

[35] E. Kurutepe and T. Sikora, "Multi-view video streaming over P2P networks with low start-up delay," in *Proc. IEEE ICIP*, 2008.

[36] Z. Chen, M. Zhang, L. Sun, and S. Yang, "Delay-guaranteed interactive multiview video streaming," in *Proc. IEEE ISCAS*, 2009.

[37] A. Mavlankar, J. Noh, P. Baccichet, and B. Girod, "Peer-to-peer multicast live video streaming with interactive virtual pan/tilt/zoom functionality," in *Proc. IEEE ICIP*, 2008.

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

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

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

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

[42] T. Hossfeld, M. Seufert, C. Sieber, T. Zinner, and P. Tran-Gia, "Identifying QoE optimal adaptation of HTTP adaptive streaming based on subjective studies," *Computer Networks*, vol. 81, pp. 320–332, 2015.

[43] J. M. Batalla, P. Krawiec, A. Beben, P. Wisniewski, and A. Chydzinski, "Adaptive video streaming: Rate and buffer on the track of minimum rebuffering," *IEEE JSAC*, vol. 34, no. 8, pp. 2154–2167, Aug. 2016.

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

[45] Y. Zhao, D. L. Eager, and M. K. Vernon, "Scalable on-demand streaming of nonlinear media," *IEEE/ACM Trans. on Networking*, vol. 15, no. 5, pp. 1149–1162, 2007.

[46] N. Carlsson, A. Mahanti, Z. Li, and D. L. Eager, "Optimized periodic broadcast of nonlinear media," *IEEE Trans. on Multimedia*, vol. 10, no. 5, pp. 871–884, 2008.

[47] R. Grigoras, V. Charvillat, and M. Douze., "Optimizing hypervideo navigation using a markov decision process approach," in *Proc. ACM Multimedia*, 2002.

[48] M. Wijnants, P. Quax, G. Alberto, R. Ruiz, W. Lamotte, J. Claes, and J.-F. Macq, "An optimized adaptive streaming framework for interactive immersive video experiences," in *Proc. BMSB*, 2015.

[49] D. Johansen, P. Halvorsen, H. Johansen, H. Riiser, C. Gurrin, B. Olstad, C. Griwodz, Å. Kvalnes, J. Hurley, and T. Kupka., "Search-based composition, streaming and playback of video archive content," *Multimedia Tools Appl.*, vol. 61, no. 2, pp. 419–445, 2012.

[50] O. Niamut, G. Thomas, E. Thomas, R. van Brandenburg, L. D'Acunto, and R. Gregory-Clarke, "Live event experiences-interactive UHDTV on mobile devices," in *Proc. IET IBC*, 2014.

[51] O. A. Niamut, A. Kochale, J. R. Hidalgo, R. Kaiser, J. Spille, J.-F. Macq, G. Kienast, O. Schreer, and B. Shirley, "Towards a format-agnostic approach for production, delivery and rendering of immersive media," in *Proc. ACM MMSys*, 2013.

[52] V. R. Gaddam, R. Langseth, H. K. Stensland, C. Griwodz, P. Halvorsen, and D. Johansen, "Scaling virtual camera services to a large number of users," in *Proc. ACM MMSys*, 2015.

[53] W. Zhang, S. Ye, B. Li, H. Zhao, and Q. Zheng, "A priority-based adaptive scheme for multi-view live streaming over HTTP," *Computer Commun.*, vol. 85, pp. 89–97, 2016.

[54] D. Seo, S. Kim, H. Park, and H. Ko, "Real-time panoramic video streaming system with overlaid interface concept for social media," *Multimedia Systems*, vol. 20, no. 6, pp. 707–719, 2014.

[55] Y. Wen, X. Zhu, J. J. P. C. Rodrigues, and C. W. Chen, "Cloud mobile media: Reflections and outlook," *IEEE Trans. on Multimedia*, vol. 16, no. 4, pp. 885–902, 2014.

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