Scalable Media Streaming to Interactive Users

Marcus Rocha, Marcelo Maia, Ítalo Cunha, Jussara Almeida, Sérgio Campos
Computer Science Department
Federal University of Minas Gerais
Av. Antônio Carlos, 6627, Pampulha
Belo Horizonte, MG, Brazil, 31270-010
{mvrocha, mmaia, cunha, jussara, scampos}@dcc.ufmg.br

ABSTRACT

Recently, a number of scalable stream sharing protocols have been proposed with the promise of great reductions in the server and network bandwidth required for delivering popular media content. Although the scalability of these protocols has been evaluated mostly for sequential user accesses, a high degree of interactivity has been observed in the accesses to several *real* media servers. Moreover, some studies have indicated that user interactivity can severely penalize the scalability of stream sharing protocols.

This paper investigates alternative mechanisms for scalable streaming to interactive users. We first identify a set of workload aspects that are determinant to the scalability of classes of streaming protocols. Using real workloads and a new interactive media workload generator, we build a rich set of realistic synthetic workloads. We evaluate Bandwidth Skimming and Patching, two state-of-the-art streaming protocols, covering, with our workloads, a larger region of the design space than previous work. Finally, we propose and evaluate five optimizations to Bandwidth Skimming, the most scalable of the two protocols. Our best optimization reduces the average server bandwidth required for interactive workloads in up to 54%, for unlimited client buffers, and 29%, if buffers are constrained to 25% of media size.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.4 [**Computer Systems Organization**]: Performance of Systems; I.6.5 [**Simulation and Modeling**]: Model Development

General Terms

Performance, Measurement, Design

Keywords

Scalable Streaming, Interactivity, Workload Model

1. INTRODUCTION

The distribution of streaming media in the Internet has faced a major challenge since its beginning: the severe limitations on scalability due to the high server and network bandwidth requirements. The conventional unicast transmission clearly does not scale, preventing media delivery to a large number of users. The efforts towards reducing bandwidth requirements have led to a plethora of new mechanisms including more bandwidth-efficient encoding methods [21], new caching strategies for reducing network and server load ([14], and references within), and scalable delivery protocols that rely on stream sharing to reduce bandwidth requirements [2, 3, 5, 7, 8, 9, 11, 12, 15].

Among the existing scalable streaming protocols, Patching [12] and Bandwidth Skimming [8, 9] are of particular interest, since they significantly reduce bandwidth requirements while still providing immediate client service. Such great achievements rely on the same principle: let clients receive data from two streams, simultaneously. The client player shows the data received in one of the streams to the user as it is received, and simultaneously buffers the data received in the earlier stream. The buffered data allows the later client to "catch up" with the one receiving the earlier stream. At this point, both streams merge and share the transmission of the rest of the file. Whereas Bandwidth Skimming allows stream merges to occur hierarchically, thus creating an arbitrarily deep merging tree, Patching restricts stream merges to a flat two-level merging tree.

Both protocols have been extensively analyzed, mainly under the assumption of synthetic *sequential* workloads, where clients request entire files, without interruption. For such workloads, both protocols have been shown to achieve significant bandwidth savings for varying client request rates. Because streams are merged hierarchically, Bandwidth Skimming scales better with request rate than Patching [9].

However, a high degree of interactivity has been observed in many real workloads [1, 4, 6, 16]. In other words, users of real media services typically *do not* access the media sequentially. Rather, they often interrupt playback, pausing, jumping (forwards or backwards), fast forwarding and rewinding. A key question that arises is, thus: *how do existing streaming protocols scale in the presence of interactive users?*

Previous efforts towards addressing this question include protocol evaluation with only a few real workloads [1, 4], and derivation of analytical bounds for general, and thus not necessarily realistic, interactive patterns [13, 18]. Though somewhat limited, these efforts reached the same conclusion: protocol scalability is severely degraded for interactive

users. The opportunities for stream sharing are reduced as requests to the same media arriving close in time at the server are often for non-overlapping segments. For very interactive users, the protocols may approach unicast, transmitting the same data multiple times, once for each request.

We are aware of only one streaming protocol optimized for interactive users, namely, the *best-effort* Patching protocol proposed in [15]. However, protocol scalability was evaluated only for simplistic (i.e., non-realistic) interactive patterns. Thus, scalable streaming to highly interactive users with immediate service is still an open and demanding issue.

This paper studies alternative mechanisms for scalable streaming to interactive users. We start by identifying a set of interactive workload aspects that are determinant to protocol scalability, aiming at facilitating the exploration of the protocol design space. Using five real media workloads and a new generator of realistic interactive media workloads, we build a rich set of synthetic workloads, falling into different application domains (entertainment, educational, audio and video) and having varying degrees of interactivity.

We then evaluate Bandwidth Skimming and Patching using our synthetic interactive workloads. Our experiments, covering a larger region of the protocol design space than previous work, show that Patching degrades much faster than Bandwidth Skimming as the degree of interactivity increases. Nevertheless, the scalability of Bandwidth Skimming still suffers for very interactive workloads. Finally, we propose and evaluate five optimizations to Bandwidth Skimming. The optimizations exploit characteristics of real workloads, such as high locality of reference [6], as well as inherent characteristics of the original protocol, and rely on buffering at the client to reduce the number of times the server has to transmit the same data.

This paper aims at reducing the average server bandwidth required to provide immediate service to interactive users. The reasons for focusing on *average* bandwidth are twofold. First, our results can be directly compared to most previous studies. Second, Tan *et al* has shown that, due to statistical fluctuations on the request rates for multiple files, the amount of bandwidth a server should be provisioned with to sustain an anticipated load is approximately the same as the *average* bandwidth required to sustain that load [19], assuming Poisson arrivals (as in [1, 6]). The focus on *streaming*, as opposed to download, stems from the greater bandwidth savings achieved by the former, when both strategies use flow sharing (i.e., multicast), even for relatively long start-up latencies [18]. Unlike scalable streaming, multicast-based download protocols approach unicast for immediate client service. Moreover, although we target *server* bandwidth, we also show some preliminary results on network bandwidth requirements. Finally, although our optimizations are described as extensions to Bandwidth Skimming, they can be applied to any hierarchical stream merging protocol [2, 5].

The main contributions of this paper are:

- Identification of interactive workload aspects that are key to the scalability of several streaming protocols.

- A generator of realistic interactive media workloads.

- A more comprehensive quantitative evaluation of Patching and Bandwidth Skimming for a rich and large set of synthetic realistic interactive workloads.

- The proposal of five protocol optimizations for further reducing bandwidth requirements for interactive users.

- A thorough quantitative evaluation of the optimized protocols for a large number of realistic workloads.

Our most relevant results show that: (1) for very interactive workloads, Bandwidth Skimming reduces the average required server bandwidth in up to 73%, compared to Patching, and (2) our best protocol optimization, which combines the efforts of three other optimizations, reduces the average server bandwidth required by the original Bandwidth Skimming in up to 54% for unlimited client buffer, and 29% for buffers constrained to only 25% of media size.

The rest of this paper is organized as follows. Section 2 discusses prior work. Our realistic interactive media workloads are presented in Section 3. Section 4 discusses the impact of interactivity on current protocols. Our protocol optimizations are presented and evaluated in Section 5. Client and network bandwidth requirements are briefly discussed in Section 6. Section 7 offers conclusions and future work.

2. BACKGROUND

This section provides an overview of previous efforts towards designing scalable streaming protocols. Existing protocols are presented in Section 2.1. Section 2.2 discusses previous work on streaming to interactive users.

2.1 Current Scalable Streaming Protocols

The conventional unicast streaming clearly does not scale with the number of users, as each user receives an independent stream. In an effort to address this problem, a number of scalable streaming protocols, which rely on stream sharing, have been proposed [2, 3, 5, 7, 8, 9, 11, 12, 15]. Two protocols of particular interest for providing immediate service while still greatly reducing server and network bandwidth requirements are Bandwidth Skimming and Patching.

Bandwidth Skimming [8, 9] is based on the *hierarchical* merging of multicast streams to reduce bandwidth requirements. The basic idea is that, upon arrival of a client request, a new (IP-level or application-level) multicast stream is created to deliver the media. In one variation of the protocol, called Closest Target, the client listens to the new multicast stream as well as to the closest previously created and still active multicast stream (its target). When the new stream has delivered all of the data that the client missed in the target stream, it is terminated, and all clients listening to the target are now "merged". The clients of the "merged" stream start listening to a new target stream, thus creating a hierarchical merging tree. However, if during the merge attempt, the target stream terminates before the later stream is ready to catch up with it, the clients listening to the later stream simply start listening to the next closest target. The data received from the target during the unsuccessful merge attempt, stored in a local *merge buffer*, are lost.

Although there are other strategies for building the hierarchical merging tree [2, 5, 8, 9], the simple Closest Target algorithm was shown to have average server and network bandwidth requirements very close to optimal [9, 22]. Furthermore, unlike other protocols, Bandwidth Skimming can be applied even if client bandwidth is less than twice the streaming rate [8]. Throughout this paper, we use the term Bandwidth Skimming to refer to the Closest Target protocol with client bandwidth equal to twice the streaming rate.

In Patching [12], the arrival of a client request triggers the creation of a multicast stream to deliver the requested media. A later arriving client joins the on-going multicast stream and receives the missed media prefix from a separate *unicast* stream. The algorithm limits the bandwidth required for unicast streams by using a *threshold window* parameter. If the missed prefix is longer than the threshold window, the server creates a new multicast stream to deliver the entire media file. Note that the merging tree created by Patching is limited to two levels. In contrast, Bandwidth Skimming may create arbitrarily deep merging trees.

Both Patching and Bandwidth Skimming have been shown to significantly reduce bandwidth requirements for sequential workloads [3, 8, 9, 11, 12]. For such workloads and Poisson arrivals, the average server bandwidth required by Patching increases with the square root of the average number of simultaneous requests received during media playback [9, 11]. In contrast, because streams are merged hierarchically, the average server bandwidth required by Bandwidth Skimming grows only logarithmically with this number [9].

2.2 Streaming to Interactive Users

A number of studies have uncovered a high degree of user interactivity in several real media workloads [1, 4, 6, 16]. Some of them have also shown that the scalability of stream sharing protocols reduces significantly for a few specific workloads [1, 4]. Moreover, recent work [13, 18] on analytically deriving the minimum required server bandwidth for immediate service to interactive users showed a great increase from previous lower-bounds for sequential workloads [9]. Those two studies are based on somewhat arbitrary user access models and *not* on real workloads (although [18] uses real workloads to motivate their access models). Thus, the accuracy of the proposed lower-bounds is still unknown.

The only streaming protocol optimized for interactive users we are aware of is the *best-effort* Patching strategy proposed in [15]. Based on buffering at the clients, the proposed protocol is a complex hybrid strategy that allows merging trees of up to three levels. It was evaluated under a very restricted model of interactivity, which was not driven from real workload observations. Moreover, restricting merging trees to only three levels may not achieve all the stream sharing an unconstrained merging tree is able to provide.

This paper contributes to previous work by: (1) providing a more comprehensive evaluation of Patching and Bandwidth Skimming for realistic workloads, with varying degrees of interactivity, and (2) proposing and evaluating a number of protocol optimizations for interactive users.

3. INTERACTIVE MEDIA WORKLOADS

A comprehensive evaluation of streaming protocols requires a set of realistic workloads that covers as much of the protocol design space as possible. Towards obtaining such workloads, Section 3.1 identifies interactive workload aspects that are key to protocol scalability. These aspects are then used, in Section 3.2, to generate a large number of realistic synthetic interactive media workloads.

3.1 Relevant Interactive Workload Aspects

This section aims at identifying a small set of interactive workload aspects that are key to the scalability of streaming protocols. We start by noting that, for sequential workloads and Poisson arrivals, the average server bandwidth required

to sustain a given load depends *only* on the normalized request rate N, expressed as the number of simultaneous requests received during media playback ($N = \lambda T$, where λ is the request rate and T is the media playback time) [9, 11].

The situation is far more complex for interactive workloads, where each user session consists of a number of requests to selected media segments. In this case, the average required server bandwidth depends not only on N, but also on a number of other parameters, such as: number of requests per session, request start position, request duration, frequency of each interaction type (i.e., pause, jump, etc), period of user inactivity between consecutive requests to the same media, and jump distance, i.e., the amount of media skipped in a jump (forwards or backwards). Moreover, these parameters may vary greatly with content type and media size, and some of them, such as request start position and request duration, are generally *not* independent [6]. These factors turn the exploration of the design space of streaming protocols for interactive users into a rather complex task.

In order to better understand the protocol design space, we must identify a smaller set of key workload aspects, derived from the aforementioned list, which capture the primary impact of the workload on protocol scalability and distinguish the behavior of alternative protocol optimizations.

We argue that the primary impact of interactive workloads on the scalability of streaming protocols, in particular stream sharing, is captured by two major factors: *temporal dispersion* and *spatial dispersion*. The term *temporal dispersion* refers to the *interactive* request rate N. Lower temporal dispersion implies a higher request rate and, thus, a larger number of successful stream merges. The term *spatial dispersion* is used to refer to the amount of media that two consecutive requests have in common, and thus that can be shared between them. In other words, it expresses the amount of overlapping media between consecutive requests. Lower spatial dispersion implies longer overlaps and, thus, more successful merges. The amount of overlap, and thus, the degree of spatial dispersion, is defined by two workload parameters: request start position and request duration[1]. Note that temporal dispersion is *not* a new definition, but rather another term for request rate, chosen so as to emphasize the impact of the two types of *dispersions* on protocol scalability. Therefore, the three parameters that determine the degree of temporal and spatial dispersion in a workload and, in turn, impact protocol scalability are *normalized request rate N*, *request start position* and *request duration*.

Figure 1 illustrates how these parameters affect temporal and spatial dispersions. It shows four workloads, each with ten requests to the same 25-minute media during the same time interval. Each line represents a request. Request arrival times are marked on the x-axis. Y-axis gives start and end positions in the media of each requested segment. Figure 1(a) shows a sequential workload, with low temporal dispersion, no spatial dispersion, and, thus, great opportunities for successful stream merges. The workload in Figure 1(b) has the same request rate $N=5$, but varying request start positions and durations, and, thus, higher spatial dispersion. There are fewer opportunities for successful merges. Compared to it, the workload in Figure 1(c) has the same

---

[1]Other workload aspects are captured by these parameters. For example, a jump distance between two requests is given by the first request end position, adding up its start position and its duration, and the second request start position.

(a) Sequential Workload   (b) Higher Spatial Disp.

(c) Lower Spatial Disp.   (d) Low Temporal Disp.
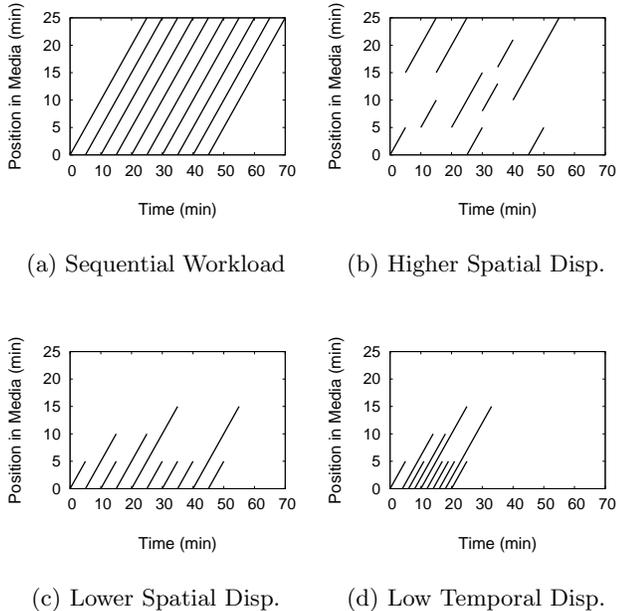
Figure 1: Dispersion in Media Workloads

temporal dispersion but lower spatial dispersion, since all requests are for a media prefix, and, thus, has a larger number of successful merges. Finally, compared to Figure 1(c), Figure 1(d) shows a workload with the same spatial dispersion but lower temporal dispersion ($N=10$), and thus, more opportunities for successful merges.

In addition to the three aforementioned workload parameters, the effectiveness of several classes of protocol optimizations may also be affected by other specific workload aspects. For instance, a *high locality of reference* in the requests within a user session, due to frequent pauses and small jump distances, implies that buffering and prefetching may significantly improve protocol scalability. More specifically, buffering previously retrieved data may benefit future requests within the same session. Thus, the *number of requests per session* is also relevant. Finally, the effectiveness of optimizations that rely on prefetching when the user is paused is affected by the *duration of inactive periods*.

Therefore, a thorough evaluation of streaming protocols for interactive workloads must consider the following determinant workload parameters: request rate, request duration, request start position, number of requests per session, locality of reference and duration of inactive periods.

3.2 Realistic Interactive Media Workloads

In [6], we performed an extensive characterization of four real media workloads falling into three different application domains: *entertainment video*, *entertainment audio* and *educational video*. The educational workload is from the eTeach media server, that delivers content at a major US university [10]. It includes variable length videos, from short announcements (under 5 minutes) to 50-60 minute lectures. The three entertainment workloads contain accesses to audio and video files, typically under 10 minutes, and were collected at two of the largest content and service providers

in Latin America, one of which is [20]. Key conclusions of that study are: (a) some workloads exhibit a high degree of user interactivity, as in [1, 4, 16], and (b) the degree of interactivity varies with content type and media size. In particular, higher interactivity is more common among users of long educational videos. More recently, we have also analyzed a three-year log of accesses to MANIC, an educational media server from another US university [17], reaching results that closely agree with those for eTeach. Collectively, this is a large and rich set of *real commercial and educational* workloads, which fall into different typical streaming application domains and enable a more comprehensive evaluation of scalable streaming media protocols than previous efforts.

The following sections describe how our five real workloads are used to facilitate the exploration of the protocol design space. Section 3.2.1 introduces a new synthetic workload generator, used to create a large set of workloads capturing key aspects of real interactive workloads. Section 3.2.2 discusses a categorization of our synthetic workloads into groups that share similar interactive patterns.

3.2.1 Synthetic Media Workload Generator

Our new synthetic media workload generator uses, as inputs, a trace of sessions to a media object and a target session arrival rate, and produces an output session trace with interactive patterns similar to the input trace. Different synthetic workloads can be built from the same input trace, varying the random seed and the target session rate. Note that the *request rate N* of a workload is a function of the session rate and the number of requests per session.

For a given input trace, our generator first builds a state-transition model, where each state represents a ten-second segment of the media object. Additional states are added to represent a *pause* and a *stop* (i.e., end of session). Fast forwarding and rewinding are rare in our workloads [6], and thus are not included. The probabilities of starting a session at each segment state as well as the transition probabilities between pairs of states are computed from the input trace. A duration is associated to each state (except the stop state). Segments that are always fully played in the input trace have duration equal to the segment size. Segments that are partially played have duration equal to the average segment playback time in the trace. The pause state has a duration equal to the average period of user inactivity in the trace. We refer to this state-transition model as a *workload profile*.

An output session trace is then created assuming session arrival process to be Poisson [1, 6]. User behavior within each session, including number of requests, jump distances and periods of inactivity, are extracted from the workload profile. The durations of pause and partially played segments are exponentially distributed with means equal to the durations of the corresponding states.

We generated workload profiles from 36 session traces, each to a different popular object in our five real workloads. These traces were selected because: (1) they contain enough requests to guarantee our workload models (one for each trace) are statistically valid, and (2) they exhibit different profiles of interactive behavior. Two typical profiles, exhibiting very different interactive patterns, are illustrated in Figure 2(a-b). The profiles show, for each request, the start and end positions in the media, ordered by start position.

We evaluated the accuracy of our synthetic workload generator in two ways. First, we compared the cumulative dis-

(a) High Interactivity
(real workload).

(b) Low Interactivity
(real workload).

(c) Validation: Average Number of
Simultaneous Streams.
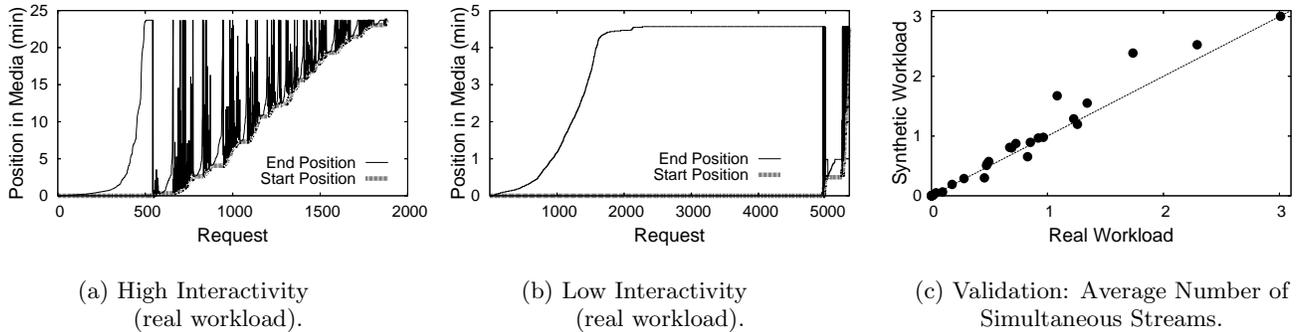
Figure 2: Workload Profiles and Workload Model Validation.

tributions of request start position and request duration for each pair of real and corresponding synthetic trace (with the same request rate N). For both parameters and all pairs of traces, the sum of squared differences between the (synthetic and real) distributions, normalized by the x-axis scale, are under 0.03. Second, we measured the average number of simultaneous streams (i.e., the average bandwidth required by unicast) for each pair of traces. Figure 2(c) shows the results, one point per pair of trace[2]. Synthetic and real measures closely agree, with errors under 27%, on average. Thus, given the complexity of real user interactive patterns, we believe our workload model does capture, at least to a first order, the key characteristics of our real workloads.

3.2.2 Categories of Interactive Workloads

Using the aspects identified as key to streaming scalability, we group our workload profiles into three categories, namely:

High Interactivity (HI): these workloads have average request duration under 20% of media duration and average request start position somewhere between 30% and 60% of media size. Typically, less than 30% of the requests start at the beginning of the media, and sessions have at least three requests. Typical HI profiles are those for long educational videos.

Low Interactivity (LI): these workloads have longer requests (at least 20% of media duration, on average), with start position heavily concentrated on the beginning of the media. There is usually less than two requests per session. Typical LI profiles are those for audio and very short videos (less than 90 seconds).

Medium Interactivity (MI): these workloads have average request duration under 20% of media duration, and start positions more concentrated on specific media points (below 30% or above 60% of media size, on average). Typical MI workloads contain sessions to entertainment videos, with less than three requests.

Out of our 36 session traces, 11 fall into the HI category, 8 into the MI category and 17 in the LI category. Figures 2(a-b) show typical profiles of HI and LI workloads. MI workloads exhibit somewhat intermediate behavior. We refer to [6] for a detailed description of our real workloads.

---

[2]The very low per-object load in the analyzed servers motivates the generation of realistic but heavier workloads.

Locality of reference and duration of inactive periods are not directly captured in our categorization. However, we note that our real and synthetic workloads exhibit a high locality of reference. On average, 33% of interactions are pause and 54% are jump backwards. Furthermore, the average jump distance is under 230 seconds, and inactive periods last for 286 seconds, on average. These results motivate some of the protocol optimizations proposed in Section 5.

The following sections evaluate current streaming protocols and new protocol optimizations for all of our synthetic workload profiles. Since workloads in the same category have similar interactive patterns, and, thus, qualitatively similar impact on protocol scalability, we show representative results for one instance of each profile. For each profile, we vary the request rate N to evaluate the impact of load intensity. Our results are average of five workloads, created from the same profile and different random seeds, and have standard deviation under 2% of the mean, in all cases.

4. IMPACT OF INTERACTIVITY ON CURRENT STREAMING PROTOCOLS

This section evaluates the scalability of two state-of-the-art streaming protocols, Patching and Bandwidth Skimming, for realistic interactive workloads. The evaluation uses our rich set of synthetic workloads and a wide range of request rates, thus covering a larger region of the protocol design space than prior work [2, 3, 5, 8, 9, 11, 12, 15].

Simulators of Patching and Bandwidth Skimming were built and validated for sequential workloads. For a range of request rates, the measured average server bandwidth required by both protocols differ from the corresponding analytical results [9, 11] by at most 11%. The simulators were then extended, with minor changes, to execute interactive workloads. One issue we had to address is how to apply the Patching threshold window to interactive requests. As in [3], we chose the less intrusive modification of applying the optimal threshold window [11] only to requests for a prefix. Thus, only streams starting at the beginning of the media can be merged. With no parameter, applying Bandwidth Skimming to interactive workloads was straightforward.

Figure 3 shows the average server bandwidth, in number of streams, required by Patching, Bandwidth Skimming and unicast for typical workload profiles, with varying degrees of interactivity and request rates. Figure 3(a) shows results for
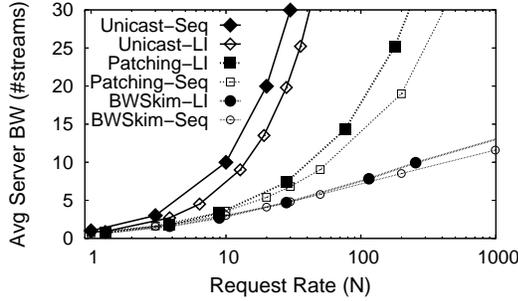
(a) Sequential and LI Workloads.
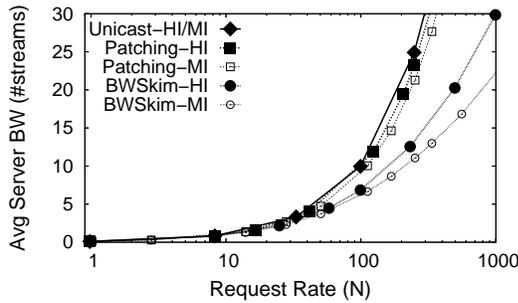
(b) HI and MI Workloads.

Figure 3: Average Server Bandwidth for Unicast, Patching and Bandwidth Skimming.

sequential and LI workloads. Figure 3(b) shows results for HI and MI workloads. Note that for a given request rate, unicast transmission requires less average server bandwidth for HI and MI workloads than for LI workloads, since the more interactive the workload is, the smaller the amount of media retrieved by each request (see Section 3.2.2).

Clearly, Bandwidth Skimming scales much better with request rate than Patching, for all workload profiles. Compared to Patching, Bandwidth Skimming reduces the average required server bandwidth for LI and HI workloads in up to 88% and 73%, respectively. The poor scalability of Patching stems from the limited opportunities for sharing in the two-level merging trees of requests for media prefixes. The frequency of such requests decreases with the degree of interactivity, and scalability degrades, approaching unicast.

Despite the superiority over Patching, the scalability of Bandwidth Skimming still suffers for very interactive HI and MI workloads, for which the high spatial dispersion causes many stream merge attempts to fail. Thus, a large amount of prefetched data are lost, and often requested again in the future. Bandwidth savings are, thus, very modest, especially for low request rates. The next section discusses several optimizations, applied to Bandwidth Skimming, that avoid data losses inherent to the original protocol, reducing the number of times the server has to retransmit the same data.

5. OPTIMIZED STREAMING PROTOCOLS FOR INTERACTIVE WORKLOADS

This section introduces our new protocol optimizations, which are described in Section 5.1 and evaluated in Section 5.2. A hybrid protocol, combining the efforts of multiple optimizations, is presented and evaluated in Section 5.3.

5.1 Description of the Protocol Optimizations

This section introduces four protocol optimizations that aim at reducing the average required server bandwidth for very interactive workloads, for which Bandwidth Skimming and Patching fail to deliver good scalability. Bandwidth Skimming, the most scalable of the two protocols, is chosen as baseline, although some of our optimizations can be applied to other stream sharing protocols as well (see below).

The common strategy in our optimizations is to save server bandwidth by reducing the number of transmissions of the same data to one or more users. Thus, our optimizations rely on buffering, at the user machine, of media segments previously received by the user or currently being delivered to other users. Future user requests to segments stored in his buffer are served locally. A client buffer exists only within a user session. Thus, its content is flushed at the end of the session. Moreover, although the buffer is presented as local to the user, managed by the client player, it could also be managed by an agent (e.g., a proxy), on behalf of the client.

Our protocol optimizations, named LOCALITY, SILENT PREFETCH, KEEP MERGE BUFFER and PRESERVE MERGING TREE, are described next. Their design is driven by characteristics of our *real* workloads, such as high locality of reference, as well as inherent characteristics of Bandwidth Skimming, namely, the hierarchical merging tree and the data losses due to unsuccessful merge attempts.

LOCALITY (LOC) buffers all media segments played by the user, exploiting the high locality of reference and the significant fraction of jump backwards found in real workloads [1, 6]. Since the buffer content is useful only for requests within the same session, LOC should perform best for very interactive workloads, with typically a large number of requests per session. Note that, under LOC, the content stored in the buffer depends only on the segments requested by the user within the session, and *not* on the request rate N.

SILENT PREFETCH (SP) exploits periods when the client is *silent*, i.e., it is not streaming from the server (and thus not using its bandwidth), to prefetch data currently being delivered to other users. The client is *silent* when the user is paused or playing from the local buffer. During such periods, the client listens to the active server stream that is the closest (i.e., delivering the closest segment) to the segment that was most recently played by the user. It also listens to the target of this stream (in the Bandwidth Skimming sense), if any. If the streams finish during the silent period, the client starts listening to the next (two) stream(s). Note that the content buffered under SP does depend on other users' requests and thus on the request rate. Nevertheless, the client has some control over this content, as it always tries to store segments that are close to the current playback position, and thus, are most likely to be accessed next.

KEEP MERGE BUFFER (KMB) stores in the buffer the segments that would otherwise be discarded during unsuccessful merge attempts. The client has little control over the buffer content, as it depends on the number, the duration and on the target of each unsuccessful merge attempt.

(a) HI Workload.   (b) MI Workload.   (c) LI Workload.
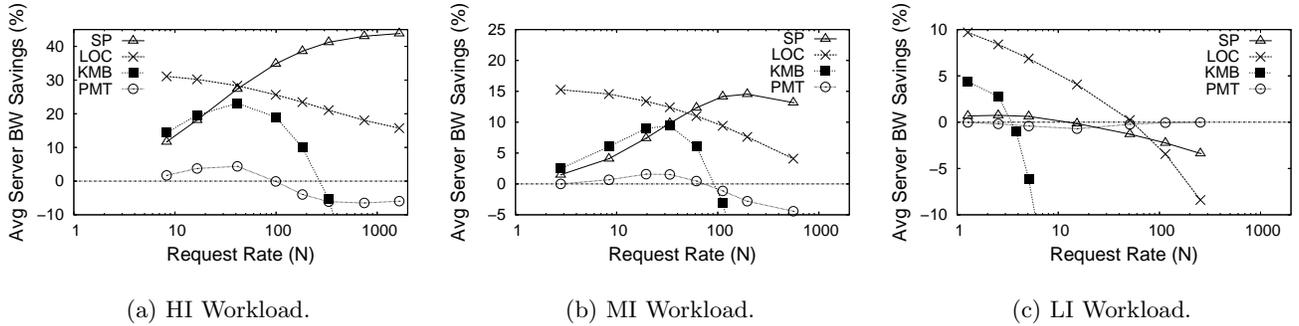
Figure 4: Average Server Bandwidth Savings Over Bandwidth Skimming (unlimited client buffers).

PRESERVE MERGING TREE (PMT) tries to prevent on-going merge attempts to be interrupted when users pause. Consider a stream X currently being listened by only one client and involved in two on-going merge attempts (i.e., it is a target of stream Y and it is also trying a merge towards stream Z). If the user receiving stream X pauses, the server extends X to allow the merge attempts to proceed and, if successful, save bandwidth. The paused client continues to listen to the extension of X, storing segments in its buffer. If the user resumes at a position nearby (high locality of reference), the next segments will be found in the buffer.

If client buffer space is constrained, our optimizations rely on the high locality of reference present in our workloads to prioritize segments that are closer (either forwards or backwards) to the segment most recently played by the user, and thus most likely to be requested in the near future. Finally, we note that our optimizations can be applied if clients have receive bandwidth less than twice the streaming rate, as the original Bandwidth Skimming [8]. Moreover, LOC and SP are general strategies that can be applied, with minor changes, to other stream sharing protocols as well.

5.2   Evaluation

This section provides a quantitative evaluation of our protocol optimizations for varying workloads and request rates. Our validated Bandwidth Skimming simulator was modified to implement the optimizations. Next, we show a few key results that capture the most relevant scalability trade-offs.

We start by noting that client buffering modifies the sequence of requests that are sent to the server, since some of the requested segments are found in the buffer. This change in the server workload impacts the effectiveness of *any protocol* based on client buffering in two opposing directions. On one hand, previously buffered data need not be retransmitted by the server in future accesses, thus reducing server bandwidth requirements. On the other hand, if buffer content is fragmented, user requests are splitted into a number of smaller and more spatially dispersed requests before being sent to the server. The higher spatial dispersion incurs a smaller number of successful merges, thus increasing server bandwidth requirements. Request fragmentation is further discussed at the end of this section.

Figure 4(a-c) shows average server bandwidth savings of each optimization over Bandwidth Skimming as a function of request rate N. Negative values represent bandwidth

penalty. The figure shows results for typical HI, MI and LI workload profiles and unlimited client buffers.

KMB, SP and PMT share the same overall behavior, for all workloads analyzed. Recall that the data buffered under these optimizations depend on the streams currently being delivered and, thus, on N. For small values of N (i.e., high temporal dispersion), there are few opportunities for buffering and thus for reducing server retransmission. Though limited, the impact of this reduction outweighs that of a more spatially dispersed server workload, yielding modest bandwidth savings. As N increases and more data is buffered, the impact of avoiding data retransmission dominates, and bandwidth savings increase. After a certain value of N, the impact of increasing spatial dispersion starts to dominate, and bandwidth savings start to decrease[3]. In fact, for very high request rates, our optimizations may require more average server bandwidth than Bandwidth Skimming. For LOC, in contrast, the impact of increasing spatial dispersion dominates as N increases, since buffer content does not depend on N, and bandwidth savings steadily decrease.

The request rate at which each optimization reaches maximum bandwidth savings depends primarily on the workload spatial dispersion. For HI and MI workloads (Figures 4(a-b)), significant savings are provided for up to high request rates. For LI workloads (Figure 4(c)), fragmentation hurts server bandwidth requirements even for small values of N.

In general, SP provides the best improvements for relatively *high request rates* ($N > 100$) and for workloads with a large number of requests per session. LOC provides the greatest savings for *low request rates* (discussed above), and for workloads with many requests per session and frequent jump backwards. In contrast, KMB provides best results for *intermediate request rates* ($10 < N < 100$), showing a fast degenerative behavior for higher rates. Since KMB buffers data gathered during unsuccessful merges, buffered segments may be far from each other, increasing buffer and request fragmentation. The design of buffer management strategies to reduce fragmentation is left for future work.

Finally, the reasons for the very modest savings (if any) provided by PMT are twofold. First, it operates only on streams received by a single client. Second, changing the merging structure is beneficial in some scenarios but may be detrimental if stream merges that would be otherwise suc-

---

[3]Bandwidth savings for SP decrease for values of N larger than those shown in Figure 4(a).
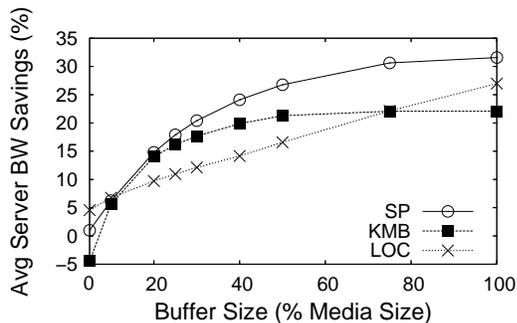
Figure 5: Impact of Buffer Size (HI profile, N=66).

(a) Unlimited Client Buffers.
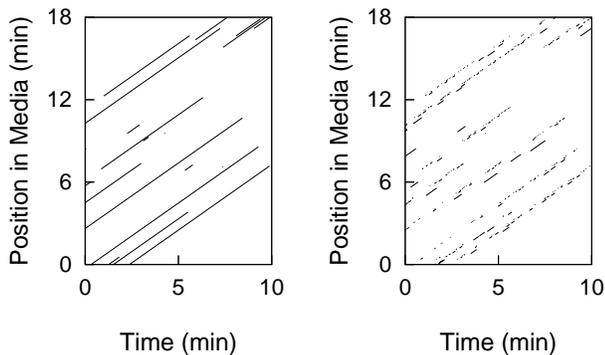
(a) Client Workload.   (b) Server Workload.

Figure 6: Request Fragmentation under KMB (HI workload, N=66, unlimited buffers).

(b) HI Workload.

Figure 7: Average Server Bandwidth Savings for Hybrid Protocol over Bandwidth Skimming.

cessful, fail in the new merging structure. We experimented with a number of protocol variants based on changing the merging structure, and none was cost-effective.

In summary, KMB, LOC and SP provide great bandwidth savings, each for different ranges of request rates. Furthermore, the most significant scalability improvements are obtained exactly for the workloads Bandwidth Skimming needs the most (i.e., very interactive workloads). The maximum bandwidth savings, provided by SP for the HI workload in Figure 4(a), is 44%. Since user requests in such workloads are already very spatially dispersed, the impact of further increasing spatial dispersion is of little relevance. Moreover, HI workloads typically have more requests per session than other profiles, favoring SP (as well as LOC).

Next, we analyze the impact of constrained client buffers. Figure 5 shows average bandwidth savings for LOC, SP and KMB for the same HI workload, N=66, and varying buffer sizes, expressed as fractions of the media size. The curves for SP and KMB show diminishing returns for large buffers. For buffers constrained to 25% (40%) of media size, KMB (SP) provides most of the benefits obtained with unlimited buffers. In contrast, LOC can always benefit from larger buffers, as long as there are a large number of requests and frequent jump backwards within typical user sessions.

In order to illustrate the phenomenon of request fragmentation, Figure 6 shows an extract of the HI workload in Figure 4(a), N=66 and unlimited buffer. Figure 6(a) shows the requests made by the users, whereas Figure 6(b) shows the requests that are sent to the server under KMB. The server workload contains a larger number of tiny and spatially dispersed requests, which are, on average, only 5% of the average user-requested segment size. Note that fragmentation is a result of client buffering in general, and may impact, to some extent, any protocol that relies on such strategy. However, depending on how buffer space is managed, some protocols are more heavily penalized. Moreover, fragmentation increases the synchronization required between server and client to handle the larger number of requests. The extra cost may be justified if bandwidth savings are significant (e.g., KMB for intermediate request rates). Otherwise, an effort has to be made to reduce request fragmentation.

5.3 Combining Multiple Optimizations

This section evaluates a hybrid strategy that combines the efforts of our best optimizations, SP, LOC and KMB, providing a unified solution that reduces server bandwidth for a larger set of configurations than any single optimization. Figure 7(a) shows average bandwidth savings for the hybrid protocol, typical HI, MI and LI workloads, and unlimited buffers. The results are envelopes of the curves for each optimization over a range of request rates (see Figures 4(a-

c)). Maximum savings are 54% and 33% for the HI and the MI workload, respectively. However, for large request rates, the degenerative behavior of KMB outweighs the benefits from SP, decreasing bandwidth savings. This result motivates the design of an online adaptive strategy that monitors the workload and reacts to changes in request rate and in interactive patterns by dynamically selecting the most cost-effective optimization(s) for each media. The design of such protocol, left for future work, could greatly benefit from the comprehensive evaluation provided in this paper.

For constrained buffers, contention for buffer space among the protocols is solved by prioritizing segments that are closer (backwards or forwards) to the current playback position. Figure 7(b) shows average server bandwidth savings as a function of buffer size, for the HI workload and varying request rates. Diminishing returns are not as clear for the hybrid protocol. Nevertheless, savings of as much as 29% are achieved with a buffer of only 25% of the media size.

6. CLIENT AND NETWORK BANDWIDTH REQUIREMENTS

Recall that the SP optimization requires the client to listen to up to *two* streams during silent periods. This reduces the available client bandwidth, and may require additional network bandwidth if a client prefetches from streams that would not be sent to its network region otherwise. This section discusses the impact of SP on client (Section 6.1) and network (Section 6.2) bandwidth requirements.

6.1 Client Bandwidth

This section compares the SP optimization, referred to as SP-Aggressive, with a protocol variant called SP-Conservative, which requires the client to listen to only *one* (the closest) stream during silent periods. Figure 8(a) shows average server bandwidth savings over Bandwidth Skimming for both variants, for a typical HI workload and unlimited buffers. Qualitative results hold for other profiles.

The pay-off for reducing the required client bandwidth by 50% is an increase of as much as 29% on the average required server bandwidth. Whereas SP-Aggressive provides bandwidth savings of as much as 44%, savings provided by SP-Conservative are limited to 30%. For buffers constrained to 25% (50%) of media size, SP-Conservative reduces average server bandwidth in up to 16% (24%), whereas SP-Aggressive provides savings of up to 19% (31%).

6.2 Network Bandwidth

This section presents a preliminary analysis of the primary trade-offs involving the server and network bandwidth required by SP. A thorough analysis is left for future work. Although we show results only for unlimited buffers, the qualitative conclusions hold for constrained buffers as well.

We use a canonical topology with a server node connected to k client sites, each representing a group of nearby users. Server and client sites are connected through a single shared link, within the server network, plus k disjoint links, each serving a distinct site. Thus, a single link is used to represent the distribution tree connecting the shared link end-point (i.e., a border router) to users within the same site. Request rate N is assumed to be homogeneously distributed among all client sites. We argue that this simple topology captures, to a first order, the dominant factors that impact network
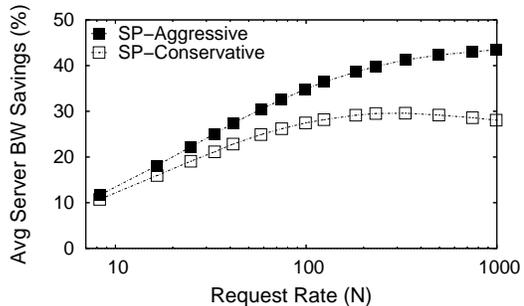
(a) Server Bandwidth (HI workload).
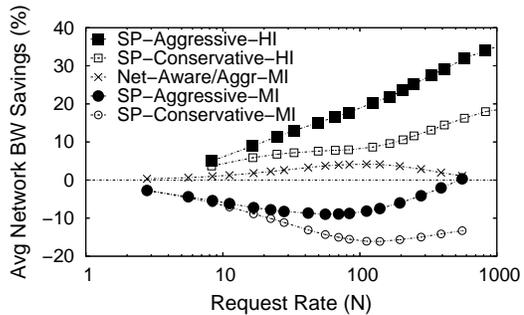
(b) Network Bandwidth (k=5).

Figure 8: Bandwidth Savings of Silent Prefetch over Bandwidth Skimming (unlimited buffers).

bandwidth requirements. Moreover, the selected topology and load distribution are the worst case with respect to how network bandwidth scales with the number of clients [22].

Figure 8(b) shows average network bandwidth savings for SP-Aggressive and SP-Conservative, for a topology with k=5 sites, a typical MI workload, and the HI workload of Figure 8(a). The average required network bandwidth, dominated by the bandwidth over the k disjoint links, is calculated as the sum of the average number of streams traversing each of these links. It depends on: (a) the number of server streams traversing the network, and (b) the number of network streams created from the sharing by multiple sites.

For the HI workload, the great server bandwidth savings (Figure 8(a)) outweight the network bandwidth increase due to inter-site stream sharing. For k=5, SP-Aggressive *decreases* the average network bandwidth required by Bandwidth Skimming in up to 37%. Greater (lower) savings are achieved with smaller (larger) values of k. For the same workload and N=500, SP-Aggressive provides network bandwidth savings of as much as 41% for k=2, and 17% for k=10. Network bandwidth savings are significant for large values of k as long as the per-site request rate is high enough to enable a significant number of intra-site stream merges.

For the MI (and LI) workload, the increase in the number of network streams dominates the modest server bandwidth savings. SP-Aggressive requires as much as 9% *more* average network bandwidth than Bandwidth Skimming. As N

increases, all three protocols experience frequent inter-site stream sharing, and the difference between them decreases.

We next evaluate average bandwidth requirements if silent prefetching is constrained to streams sent to the same site. We call this strategy Network-Aware SP. Note that the merges performed during active periods are still network-oblivious, as constraining such merges has little impact on network bandwidth [22]. Figure 8(b) shows results for Network-Aware SP-Aggressive, for the same MI workload, and k=5. In contrast to the network-oblivious SP-Aggressive, network-awareness provides limited network bandwidth savings (up to 4%), at the cost of a modest 5% maximum server bandwidth savings (omitted). Thus, for workloads with low interactivity, network-awareness provides only marginal gains over Bandwidth Skimming. For more interactive workloads (omitted), on the other hand, network and server bandwidth savings are as high as 35% and 29%, respectively.

7. CONCLUSIONS AND FUTURE WORK

This paper investigates new strategies for scalable streaming to interactive users. We devise a framework that helps understanding which interactive workload aspects affect the scalability of classes of streaming protocols, and use it to build a set of realistic workloads, and to guide the evaluation of the current Bandwidth Skimming and Patching protocols, as well as five new protocol optimizations.

Our results show that it is possible to greatly improve the scalability of Bandwidth Skimming, the best of the two current protocols, for varying interactive workloads and request rates. Moreover, the greatest benefits are obtained for very interactive workloads, for which the original protocol fails to scale well. Our best optimization reduces average server bandwidth requirements in up to 54%, for unlimited client buffers, and 29%, for buffers constrained to 25% of media size. We uncover the impact of buffering on request fragmentation and protocol scalability, and provide initial results on client and network bandwidth trade-offs.

Future work includes designing efficient buffer management policies to reduce fragmentation, as well as new protocols, including an adaptive strategy that dynamically applies alternative optimizations based on the current workload.

8. ACKNOWLEDGMENTS

9. REFERENCES

[1] J. Almeida, J. Krueger, D. Eager, and M. Vernon. Analysis of Educational Media Server Workloads. In *Proc. NOSSDAV*, Port Jefferson, NY, June 2001.

[2] A. Bar-Noy and R. Ladner. Competitive On-Line Stream Merging Algorithms for Media-on-Demand. *Journal of Algorithms*, 48(1):59–90, Aug. 2003.

[3] M. Bradshaw, B. Wang, S. Sen, L. Gao, J. Kurose, P. Shenoy, and D. Towsley. Periodic Broadcast and Patching Services- Implementation, Measurement and Analysis in an Internet Streaming Video Testbed. *Multimedia Systems*, 9(1):78–93, July 2003.

[4] M. Chesire, A. Wolman, G. Voelker, and H. Levy. Measurement and Analysis of a Streaming Media Workload. In *Proc. Symp. on Internet Technologies and Systems*, San Francisco, CA, Mar. 2001.

[5] E. Coffman, P. Momcilovic, and P. Jelenkovic. The Dyadic Stream Merging Algorithm. *Journal of Algorithms*, 43(1):120–137, Apr. 2002.

[6] C. Costa, I. Cunha, A. Borges, C. Ramos, M. Rocha, J. Almeida, and B. Ribeiro-Neto. Analyzing Client Interactivity in Streaming Media. In *Proc. World Wide Web Conference*, New York, NY, May 2004.

[7] D. Eager and M. Vernon. Dynamic Skyscraper Broadcasts for Video-on-Demand. In *Proc. International Workshop on Advances in Multimedia Information Systems*, Istanbul, Turkey, Sep. 1998.

[8] D. Eager, M. Vernon, and J. Zahorjan. Bandwidth Skimming: A Technique for Cost-Effective Video on Demand. In *Proc. Multimedia Computing and Networking*, San Jose, CA, Jan. 2000.

[9] D. Eager, M. Vernon, and J. Zahorjan. Minimizing Bandwidth Requirements for On-Demand Data Delivery. *IEEE Transactions on Knowledge and Data Engineering*, 13(5):742–757, Sep. 2001.

[10] eTeach. http://eteach.cs.wisc.edu/index.html.

[11] L. Gao and D. Towsley. Supplying Instantaneous Video-on-Demand Services Using Controlled Multicast. In *Proc. IEEE Multimedia Computing Systems*, Florence, Italy, June 1999.

[12] K. Hua, Y. Cai, and S. Sheu. Patching: A Multicast Technique for True Video-on-Demand Services. In *Proc. ACM MULTIMEDIA*, Bristol, UK, Sep. 1998.

[13] S. Jin and A. Bestravros. Scalability of Multicast Delivery for Non-sequential Streaming Access. In *Proc. SIGMETRICS*, Marina Del Rey, CA, June 2002.

[14] J. Liu and J. Xu. Proxy Caching for Media Streaming over the Internet. *IEEE Communications Magazine*, 42(8):88–94, Aug. 2004.

[15] H. Ma, K. Shin, and W. Wu. Best-effort Patching for Multicast True VoD Service. *Multimedia Tools Applications*, 26(1):101–122, May 2005.

[16] J. Padhye and J. Kurose. An Empirical Study of Client Interactions with a Continuous-Media Courseware Server. In *Proc. NOSSDAV*, Cambridge, UK, July 1998.

[17] RIPPLES/MANIC. http://manic.cs.umass.edu.

[18] H. Tan, D. Eager, and M. Vernon. Delimiting the Range of Effectiveness of Scalable On-Demand Streaming. In *Proc. International Symposium on Computer Performance Modeling and Evaluation*, Rome, Italy, Sep. 2002.

[19] H. Tan, D. Eager, M. Vernon, and H. Guo. Quality of Service Evaluations of Multicast Streaming Protocols. In *Proc. ACM SIGMETRICS*, Marina Del Rey, CA, June 2002.

[20] Universo Online. http://www.uol.com.br.

[21] B. Vickers, C. Albuquerque, and T. Suda. Source-adaptive Multilayered Multicast Algorithms for Real-time Video Distribution. *IEEE/ACM Transactions on Networking*, 8(6):720–733, Dec. 2000.

[22] Y. Zhao, D. Eager, and M. Vernon. Network Bandwidth Requirements for Scalable On-Demand Streaming. In *Proc. IEEE INFOCOM*, New York, NY, June 2002.