Scalable On-Demand Media Streaming with Packet Loss Recovery *

Anirban Mahanti Derek L. Eager Mary K. Vernon David Sundaram-Stukel

Dept. of Computer Science
University of Saskatchewan
Saskatoon, SK S7N 5A9 Canada
{mahanti, eager}@cs.usask.ca

Computer Sciences Dept.
University of Wisconsin
Madison, WI 53706-1685 USA
{vernon, sundaram}@cs.wisc.edu

ABSTRACT

Inspired by recent techniques for reliable bulk data distribution, this paper develops scalable protocols for reliable on-demand delivery of streaming media. Models are developed that quantify the best possible scalability for given client characteristics. The results of the models are used to guide the design and assess the performance of the proposed streaming techniques. The new protocols, RPB and RBS, are relatively simple to implement and achieve nearly the best possible scalability and efficiency for a given set of client characteristics and desirable/feasible media quality.

1. INTRODUCTION

An important problem for a number of existing and future Internet applications is that of delivering streaming media on-demand, in a *scalable* and *reliable* manner, to potentially large numbers of concurrent clients that receive the data over lossy and possibly heterogeneous channels. This problem has been addressed effectively for bulk data distribution [28, 31, 5], such as large software updates. However, adequate solutions do not currently exist for streaming media content, as would be required in applications such as video-on-demand.

The *digital fountain* [5] approach [28, 31, 5] is designed to deliver bulk data over channels that have significant packet loss, including IP multicast on the Internet, satellite transmission, and wireless transmission channels. The approach uses *erasure codes* to construct a stream of packets, such that a receiver can reconstruct the object from any subset of the packets of size equal to or just slightly greater than the number of packets in the source data. Thus, each client can begin listening to the (multicast) stream at a time of their own choosing, and simply keep listening until the required number of packets have been correctly received. The method is fully scalable because the required server transmission bandwidth is independent of the number of clients actively acquiring the data. The method is efficient because (1) no feedback channels are required for clients to recover lost packets, (2) the amount of data each client needs to receive in order to obtain the full object is (nearly) minimal, and (3) the amount of processing time required for a client to reconstruct the original data is small.

The digital fountain described above is inapplicable to streaming media, however, since in general a client would be unable to reconstruct the portion of the object at which playback is to begin until after receiving most or all of the data needed to reconstruct the entire object. Other recent approaches for reliable *live* or *scheduled* broadcast delivery of streaming media [33, 13, 7, 23, 4, 21, 30, 9] do not address the problem of providing scalable *on-demand* delivery.

Conversely, recently proposed protocols for scalable on-demand media streaming, such as *periodic broadcast* protocols [32, 1, 17, 20, 14, 25, 16], *patching* [8, 18, 6, 15, 29], and *bandwidth skimming* [12], do not address the issue of providing reliable delivery over lossy channels. Moreover, it is not straightforward to extend these streaming protocols to include redundant data for recovering lost packets. For example, in most protocols, each transmitted media packet is needed for (nearly) immediate playback by at least one of the clients receiving the packet. In all other cases, and in all previous periodic broadcast protocols, clients must be able to receive multiple streams that have an aggregate transmission rate equal to two or more times the minimum rate required for real-time playback. Such aggregate transmission rates may be infeasible for a given desired media quality and transmission path. Even if the transmission rates are feasible, they may be suboptimal for environments where higher transmission rate implies higher probability of packet loss (e.g., due to congestion) in the transmission path. When applying these scalable streaming methods in environments where packet loss is relatively rare, local error concealment (e.g., interpolating lost video frames from frames that are received correctly) may be adequate. However, in environments that have frequent and bursty packet loss, local error concealment is inadequate for many applications [26].

This paper develops new scalable periodic broadcast and bandwidth skimming protocols for reliable, on-demand delivery of streaming media content over lossy and possibly

*This work was partially supported by the Natural Sciences and Engineering Research Council of Canada under Grant OGP-0000264 and by the National Science Foundation under Grant CCR 9975044.

SIGCOMM'01, August 27-31, 2001, San Diego, California, USA..

heterogeneous channels. First, models are developed that quantify the best possible scalability for given client characteristics. The models are used to guide the design and assess the performance of the proposed new protocols. The new protocols are relatively simple to implement and achieve nearly the best possible scalability and efficiency for a given set of client characteristics and desirable/feasible media quality. The paper also proposes using the new protocols to allow clients who arrive late to a live or scheduled multicast to request the earlier media content that they missed, with only a very modest increase in server bandwidth.

The new Reliable Periodic Broadcast (RPB) protocols are fully scalable, include efficient transmission of redundant data for clients with heterogeneous packet loss rates, require minimal client feedback, and have a tunable latency for beginning the media playback. The RPB protocols also assume that the maximum aggregate transmission rate to any client is equal to a parameter that can be set to a small percentage (e.g., 25%) greater than the minimum transmission rate required for real-time playback.

The new Reliable Bandwidth Skimming (RBS) protocols require limited feedback from the clients (i.e., primarily the client requests that initiate new streams) and are not as efficient as the RPB protocols with respect to the amount of data that is received by a client. However, the RBS protocols have minimal startup latency for playback, fully support interactive client requests, and operate more efficiently than the RPB protocols when fewer clients request the media.

The remainder of the paper is organized as follows. Section 2 reviews previous periodic broadcast and bandwidth skimming protocols, and previously developed lower bounds on the required server bandwidth for each of these two classes of protocols. Section 3 compares alternative packet loss recovery strategies, both qualitatively, and through the derivation of simple lower bounds on the required server bandwidth. Sections 4 and 5 develop the proposed new RPB and RBS protocols, respectively, and provide qualitative and quantitative assessments of the new protocols. Conclusions are presented in Section 6.

2. BACKGROUND

The new protocols proposed in this paper use the previously developed concepts of periodic broadcast and bandwidth skimming for scalable on-demand delivery of streaming media. Previous periodic broadcast and bandwidth skimming protocols are briefly reviewed in Sections 2.1 and 2.2, respectively. Section 2.3 reviews a previous analytic model [3, 10] that provides a lower bound on the server bandwidth required for each class of protocol. This model serves as a basis for the new bounds that are developed in Section 3 for evaluating the bandwidth requirements of alternative strategies for packet loss recovery in the context of scalable on-demand streaming. Finally, Section 2.4 summarizes the goals of the new protocols developed in Sections 4 and 5 of the paper.

2.1 Previous Periodic Broadcast Protocols

Previous periodic broadcast schemes divide a media file into K segments, with a progression of relative segment lengths, $l_1, l_2, ..., l_K$. In the simplest case, each segment is repeatedly broadcast (or multicast) on its own channel (e.g., multicast group) at the media playback rate. Although a number of such protocols have been proposed, for the purposes of this paper it suffices to describe just a single example. The *skyscraper* broadcast protocol [17] has segment length progression $1, 2, 2, 5, 5, 12, 12, 25, 25, ...$ and a broadcast schedule as depicted in Figure 1 for $K = 6$.

A client arriving at an arbitrary point in time obtains a schedule for tuning in to each channel to receive each of the segments, starting at the beginning of the next segment 1 broadcast on channel 1. For example, a client who arrives at the time indicated by the arrow is given the schedule of shaded segment broadcasts. The periodic broadcast schedule ensures that, for any segment 1 broadcast, the client can receive each other media segment at or before the time it is needed for playback. Since segments are increasing in size, clients that initially start listening to different segment 1 broadcasts often listen to the same broadcasts of later segments. The small first segment permits low startup latencies while the larger later segments keep the total number of channels needed for the broadcast small.

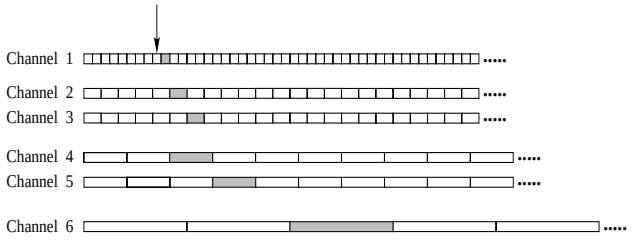


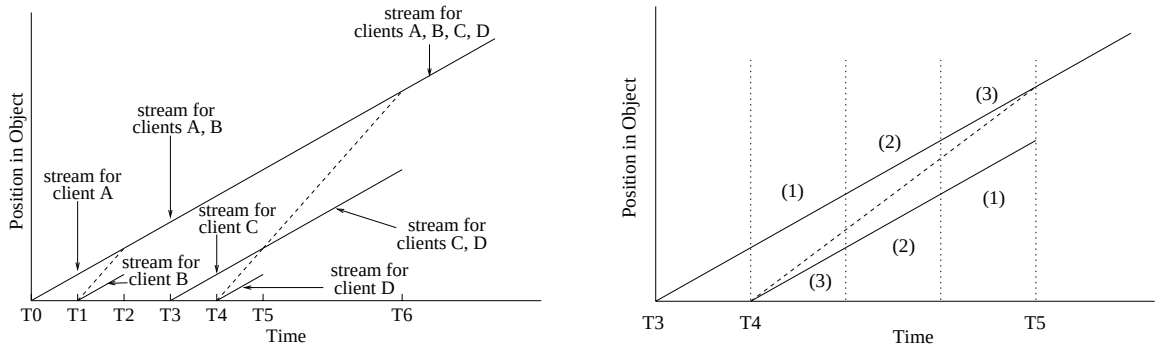
Figure 1: Skyscraper Broadcasts ($K = 6$)

Skyscraper systems have the key property that clients listen to at most two playback rate channels concurrently. Clients must have buffer space equal to l_K to store data until it is needed for playback. Furthermore, the maximum time a client waits to begin receiving a media stream of duration T is $T/\sum_{k=1}^{K} l_k$; e.g., if $K = 10$, the maximum latency is $0.007T$.

Other periodic broadcast protocols have been devised that reduce the amount of server bandwidth required for a given maximum startup latency, often at the cost of increasing the required aggregate transmission rate to each client and in some cases also more complex broadcast schedules [32, 1, 20, 14, 25, 16]. For example, the recent work by Hu [16] derives the optimal segment sizes and transmission rates assuming (1) a fixed number of segments, (2) each segment is completely received prior to beginning its playout, and (3) each client can receive *all segments concurrently*. No algorithm is presented for determining segment sizes and transmission rates in other cases. For the special case that clients can listen to at most two play-rate streams, Hu briefly suggests a Fibonacci series segment size progression with holes in the transmission schedule. Hu also briefly discusses how periodic broadcast protocols can support a limited form of client fast-forward requests.

2.2 Previous Bandwidth Skimming Protocols

Bandwidth skimming protocols [12] initiate a new multicast transmission of the media file for each new client request. In the simplest case, each client also listens to the closest earlier stream that is still active [10, 11], so that its own stream can terminate after transmitting the data that was missed in the earlier stream, as illustrated in Figure 2(a). In the figure, clients A through D request the media object at times T0, T1, T3, and T4, respectively. At



(a) Hierarchical Multicast Stream Merging (b) Partition

Figure 2: Bandwidth Skimming Example

Table 1: Notation for Scalability Bounds

Symbol	Definition
λ	average client request rate for a media object
T	media object playback duration
N	average number of requests for the object that arrive during a period of length T ($N = \lambda T$)
d	maximum start-up delay for object playback
B	required server bandwidth (in units of the object playback bit rate)

time T4, client D listens to the stream that starts at T4 as well as the stream that was initiated for client C at time T3. At time T5, the stream for client D can be terminated, and clients C and D are "merged". When clients merge, they begin listening to the closest earlier stream that is still active, and so on.

An important feature of the bandwidth skimming protocols is that the hierarchical merging illustrated in Figure 2(a) can be implemented when the aggregate transmission rate to the client is less than twice the media playback rate. For example, in one version of the protocol (called *Partition* in [12]), each stream is transmitted at the playback rate, but on k channels, where k is a parameter of the protocol. Each channel carries $1/k$ of the stream data using a deterministic fine-grained interleaving of the data packets. Figure 2(b) illustrates how client D merges with client C when each stream is transmitted on three channels (i.e., $k = 3$), and client D can listen to at most four channels for a maximum client data rate of 1.33 times the media playback rate. Client D goes through three distinct periods between arriving at time T4 and merging with client C at time T5. During each period, client D listens to the number of channels from each stream indicated in parentheses near the stream. For example, in the first period, client D listens to one of the channels of client C's stream. In the second period, client D has already received the data that will be delivered on one of its own channels, so client D listens to two channels of its own stream and two channels of client C's stream.

Results in [12, 10] show that the server transmission bandwidth used for the bandwidth skimming protocols increases only logarithmically (with a small constant factor) as the client request rate increases. Those results show that bandwidth skimming is more efficient than periodic broadcast protocols at low to moderate client request rates (e.g., under 100 requests per time T). The results also show that even with client data rate only 1.25 times the media playback rate, bandwidth skimming yields similar or (for high request rates) substantially better performance than the optimized patching technique [6, 15], which requires client data rate equal to twice the media playback rate. Furthermore, in contrast to both optimized patching and periodic broadcast, bandwidth skimming naturally allows each client to start at an arbitrary point in the media stream, and thus only very simple extensions to the protocol are required to support client interactive requests including general "fast forward" requests [10]. Adding a maximum startup delay to the bandwidth skimming protocol can further decrease server transmission bandwidth, as shown in [12].

2.3 Maximum Achievable Scalability

In [10] a tight lower bound is derived on the required server bandwidth[1] for *any* protocol that provides immediate on-demand streaming of multimedia content, with no packet loss recovery. Assuming the notation in Table 1, the lower bound is as follows:

$$B_{minimum}^{immed-service} = \int_0^T \frac{dx}{x + \frac{1}{\lambda}} = \ln(N+1). \qquad (1)$$

The bound was derived by considering a small portion of the object at some arbitrary time offset x. For an arbitrary client request that arrives at time t, this portion of the object must be delivered no later than time $t + x$. If the portion is multicast at time $t + x$, then (at best) those clients that request the file between time t and $t + x$, can receive the same multicast. Since the average time from $t + x$ until the next request for the object is $1/\lambda$, the minimum frequency of multicasts of the portion at time offset x is $1/(x + 1/\lambda)$, which yields the above bound. Note that the required server

[1]The "required server bandwidth" for an object is defined as the average server bandwidth used by the specified protocol. This bandwidth is measured in units of the media play rate.

bandwidth for delivery of multiple objects can be derived by summing the bandwidth required for each object, weighted by the relative playback rate.

The bound in equation (1) assumes that the client data rate is unbounded and request arrivals are Poisson as measured in [2]. As described in [10], the latter assumption can be relaxed to cover a wide class of arrival processes (including those with heavy-tailed interarrival time distributions), yielding a similar analytic result with difference bounded by a constant independent of λ. Note also that, as illustrated in [12], the Poisson arrival assumption yields conservative server bandwidth estimates for heavy tailed interarrival time distributions, since greater burstiness in the arrival process causes clients to be merged more quickly.

The bound in equation (1) can be extended by adding a start-up delay d to the minimum time between multicasts of the portion at position x (i.e., add d to the denominator of the integrated function). For periodic broadcast schemes, which accommodate arbitrary (i.e., essentially infinite) client arrival rate, this yields the following lower bound on required server bandwidth, as shown in [3]:

$$B_{minimum}^{periodic-bcast} = \int_0^T \frac{dx}{x+d} = \ln(\frac{T}{d}+1). \quad (2)$$

2.4 Goals of the New Streaming Protocols

The key goals of the new delivery protocols developed in Sections 4 and 5 are:

- Convenient: Clients can begin playing the requested media content after a tunably small start-up latency.
- Tolerant: The protocol should tolerate clients with heterogeneous packet loss rates and transmission path data rates.
- Reliable: Clients that have a packet loss rate up to a tunable value should be able to reconstruct each media packet prior to its play point.
- Efficient: The protocol should require minimal client feedback, and, if possible, the total amount of data each client receives should be minimal.
- Scalable: The protocol should allow streaming of content on-demand to large numbers of concurrent clients.

3. LOSS RECOVERY STRATEGIES

This section compares three basic strategies for packet loss recovery for on-demand media streaming, namely: unicast retransmission of lost packets, multicast retransmission of lost packets, and multicast transmission of redundant data that is computed using erasure codes. Server-based recovery is assumed; at the cost of additional infrastructure and complexity, distributed recovery architectures are also possible and have been extensively explored [21].

The conclusion that is drawn from the comparisons below is consistent with conclusions drawn for the reliable single-stream (e.g., live) multicast setting [23]. That is, erasure codes provide a better solution. However, the analysis quantifies the benefits of erasure codes with respect to the server load imposed by error recovery for scalable on-demand streaming. In the process, new fundamental lower bounds on required server bandwidth are derived that will be applied later in the paper.

3.1 Qualitative Discussion

The three basic approaches can be compared qualitatively along at least three dimensions, namely: implementation complexity, start-up delay, and scalability. Regarding implementation complexity, the use of erasure codes entails the overhead of encoding and decoding the data, but approaches based on retransmission must handle unpredictable retransmission requests and feedback implosion (e.g., [21]).

The *start-up delay* is defined as the time from when a client requests a media object, until the client can begin playback. A retransmission-based approach must have sufficient start-up delay to allow for clients to request retransmissions and for retransmitted data to be successfully received prior to its play point. In an approach based on erasure codes, media data is coded/decoded in coarse grained *blocks*, and, further, these blocks may be interleaved during transmission so as to reduce sensitivity to burst losses. The start-up delay must be sufficient to allow time for each entire block to be received and decoded at the client. In either case, the start-up delay requirement is additive with other start-up delay components, such as that required to deal with network jitter. One might expect total start-up delays to be of similar magnitude in both approaches, but the delay may be more easily estimated for an approach that uses erasure codes and a given interleaved transmission schedule.

Scalability concerns how the server bandwidth required to reliably deliver an object on-demand must increase as a function of the object request rate and client packet loss rates. For multicast streaming, different clients experience different packet losses, and the alternative packet loss recovery schemes differ with respect to effective sharing of redundant data transmissions, and thus with respect to server bandwidth requirements. Multicast retransmissions require the server to resend only one copy of data that multiple clients have not received, but other clients may receive more data than they need to recover from their own losses. Multicast transmission of redundant data computed using erasure codes permits a single redundant packet to repair different losses for different clients, which leads to the qualitative notion that this approach may be more scalable than the retransmission-based approaches.

3.2 Quantitative Scalability Bounds

The lower bounds on required server bandwidth for any protocol that provides reliable on-demand streaming using a specific packet recovery strategy build on the bounds for no packet loss recovery reviewed in Section 2.3. The simplest lower bound is for the case that erasure codes are used to recover from packet loss. In this case, each client must receive an amount of (source and redundant) data that is at least equal to the size of the object. If the average client packet loss probability is p, in the best case each client has average packet loss probability equal to p, and in this case the server must transmit an amount of data per unit of time (on average) that is a factor of $\frac{1}{1-p}$ greater than when packet loss recovery is not performed (since on average $1-p$ of the packets will be received). Thus,

$$B_{minimum}^{immed-service,erasurecodes} = \frac{1}{1-p}\ln(N+1), \quad (3)$$

and

$$B_{minimum}^{periodic-bcast,erasurecodes} = \frac{1}{1-p}\ln(\frac{T}{d}+1). \quad (4)$$

The above bounds are used to evaluate the proposed new protocols in Sections 4 and 5.

For unicast retransmission of lost packets, if the object playback duration is T minutes, the average amount of data that is retransmitted per client, measured in playback minutes, is equal to $\frac{pT}{1-p}$. Given a client request rate of λ, the server bandwidth required just for the retransmitted data is $\lambda\frac{pT}{1-p}$. Using $N = \lambda T$ and the minimal server bandwidth needed for the original packet transmissions from the bound in equation (1), yields

$$B_{minimum}^{immed-service,unicast-retrans} = \ln(N+1) + \frac{p}{1-p}N. \quad (5)$$

The derivation of the lower bound on required server bandwidth in the case of multicast retransmissions of lost packets is more complex and is provided in Appendix A. The bound is not obtained in closed form, but can be solved numerically for particular values of N and p. The bound for multicast retransmissions assumes that each packet is lost with independent probability p, although generalizations are possible.

The above bounds for unicast retransmissions of lost packets and for multicast transmission of redundant data using erasure codes can be generalized for non-Poisson request arrival processes as described in [10] for the previous bound in equation (1). The bound for multicast retransmissions is more complex, and at present a generalization for non-Poisson arrival processes has not been formulated.

3.3 Numerical Results

Figure 3 presents the bounds derived above as functions of the normalized client request rate N, for 15% average packet loss probability (p). Smaller but similar and still significant differences in the bounds are observed for lower values of p.

Results in Sections 4 and 5 will show that the lower bound for packet loss recovery using erasure codes can be closely approached by actual streaming protocols, given that each client has cumulative packet loss less than or equal to p for each media segment received. The results in Figure 3 show that even a "perfect" retransmission-based recovery strategy would require more server bandwidth. Given the implementation difficulties for retransmission based approaches, the new reliable broadcast protocols developed next use the erasure coding strategy.

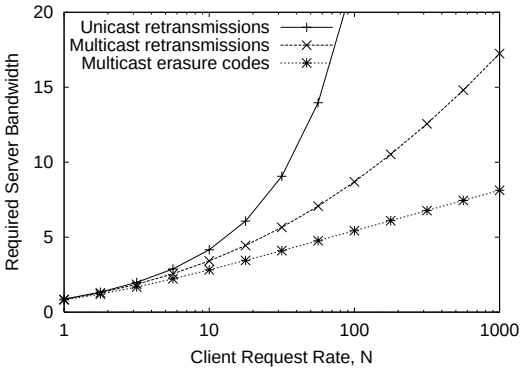


Figure 3: $B_{minimum}$ for Immediate Service & Packet Loss Recovery (15% Packet Loss)

4. RELIABLE PERIODIC BROADCAST

In this section we develop optimized periodic broadcast protocols that (1) assume a maximum aggregate transmission rate to any given client that is a tunable parameter, n, which may be less than (or greater than) twice the media play rate, and (2) enable clients with heterogeneous loss probability to recover from packet loss. The notation used in developing the RPB protocols is given in Table 2. To deal with the fundamental challenges involved in designing such optimized protocols, we proceed in four stages.

Section 4.1 develops a family of Optimized Periodic Broadcast (Optimized PB) protocols that do not support packet loss recovery, but are optimized under the constraint that clients receive each segment entirely before playing the beginning of the segment. For a specified segment streaming rate (r) and maximum number of streams that clients can listen to concurrently (s), the Optimized PB protocols allow each client arriving at an arbitrary time to immediately begin receiving the first s segments of the media object. Furthermore, under the stated constraints and assuming each client begins playing the object (nearly) immediately after receiving the first segment, the Optimized PB protocols have the maximum possible segment size increases. These protocols thus have the minimum possible start-up delay for a given total server bandwidth, or the minimum possible server bandwidth for a given start-up delay, under the stated constraints. Performance of the new protocols is compared against the previous skyscraper protocol.

Section 4.2 extends the Optimized PB protocols to create a family of basic Reliable Periodic Broadcast (Basic RPB) protocols. These protocols transmit data that has been encoded using erasure codes, enabling each client to reconstruct each segment k prior to the time the beginning of the segment needs to be played, assuming the fraction of packets lost in transmitting segments $1-k$ to the client is not greater than a tunable parameter p which has the same value for each segment. The Basic RPB protocols (1) allow each client to immediately begin receiving the first s segments of the media, and (2) have maximum segment size increases for the given values of r, s, and p. Appendix B contains an asymptotic analysis of the new Basic PB protocols which shows that the required server bandwidth can approach the lower bound for reliable delivery using erasure codes, as provided in equation (4). The analysis also suggests that the protocols can approach the minimum possible required server bandwidth under any client rate constraint.

Section 4.3 generalizes the Basic RPB protocols to allow each segment to have a different associated cumulative loss protection. These RPB protocols address the impact of bursty packet losses on reliable reconstruction of earlier segments, and allow the estimation of cumulative packet loss rate to be less conservative for later segments.

Heterogeneous client packet loss probabilities and situations that cause packet loss to exceed the specified upper bound on packet loss are addressed in Section 4.4.

4.1 Optimized PB Protocols

The proposed new family of Optimized PB protocols assumes that (1) packet loss in any given transmission path can be adequately addressed using local error concealment, and (2) each segment must be entirely received before the beginning of the segment is played. To minimize start-up delay, the first segment is repeatedly transmitted on a given

Table 2: Parameters of the new Optimized and Reliable Periodic Broadcast Protocols

Symbol	Definition
K	total number of segments
r	segment transmission rate (in units of the object playback bit rate)
s	assumed maximum number of streams that clients listen to concurrently
n	assumed maximum aggregate transmission rate to a client (in units of the object play rate), $n = s \times r > 1$
l_k	length (playback duration) of the kth segment (relative to the length of segment 1)

multicast channel. Thus, a client arriving at an arbitrary point in time can immediately begin listening to the channel for a period of time equal to the time it takes to transmit the segment, as illustrated by the shaded portion of channel 1 in Figure 4. Also to minimize start-up delay, the client will begin playing the first segment when it is fully received. The segment size progression is designed so that the client will receive each other segment in its entirety just in time to begin playing the segment. Each segment is repeatedly transmitted on its own multicast channel, so that each client arriving at an arbitrary point in time will be able to fully receive and begin playing each media segment on time.

To derive the maximum possible segment size increases, first consider the case that each segment is delivered at the object playback rate[2] (i.e., $r = 1$) and multicast join operations have zero latency. In this case, if s is the assumed maximum number of segment transmissions a client can simultaneously listen to, then each segment k, $1 < k \leq s$, has maximum length equal to the time to receive segment 1 plus the time to play each earlier segment (i.e., segments 1 through $k-1$). For segment $k > s$, the client will begin receiving segment k at the time that segment $k-s$ is just received and starts playing. The client must finish receiving segment k when segment $k-1$ finishes playing. Thus, measuring segment lengths relative to the first segment length, we have $l_1 = 1$ and the following maximum sizes for other segments:

$$l_k = \begin{cases} l_1 + \sum_{j=1}^{k-1} l_j & 1 < k \leq s \\ \sum_{j=k-s}^{k-1} l_j & k > s. \end{cases} \quad (6)$$

For a given number of server streams, K, used to multicast the object, the total server bandwidth used is $B = r \times K$ and the (deterministic) client start-up delay is equal to $\frac{T}{r \sum l_k}$.

Figure 4 illustrates the segment sizes and a transmission schedule for the Optimized PB protocol with parameters $K = 6$, $r = 1$, and $s = 2$. Note that (1) if $r = 1$ and $s = 2$, the segment length progression is the Fibonacci series, and as with other Optimized PB parameter settings, the segment transmission schedule has no holes; (2) any alignment of transmissions between any two channels is valid, and (3) two clients arriving at different points in time generally tune into each segment multicast stream at different times.

[2]For simplicity, the development of the segment sizes assumes constant bit rate content or a fully smoothed variable bit rate (VBR) media stream with the requisite additional start-up delay (if any). The segment sizes can be further optimized for VBR content that hasn't been (fully) smoothed, in which case each segment would be fully smoothed and delivered in a constant bit rate stream, but calculation of the optimized segment transmission rates and sizes is substantially more complex for this case.

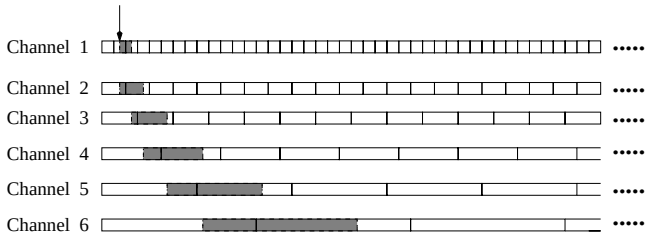


Figure 4: Example Optimized PB Protocol ($K = 6$, $r = 1$, $s = 2$)

Generalizing the Optimized PB protocols for segment transmission rate $r \leq 1$ provides the capability to assume that the maximum aggregate transmission rate to each client, $n = s \times r > 1$, is less than twice the minimum transmission rate required for real-time playback. Note that this allows more of the achievable transmission rate to a client to be used for delivering higher quality media content. Furthermore, as will be illustrated below, for a fixed value of n and total server transmission bandwidth (B), a lower value of r yields a lower start-up delay (but a larger number of server multicast transmission streams, $K = B/r$).

Noting that the time to download each segment is equal to $1/r$ times the segment length, the maximum relative segments sizes are easily generalized for $r \leq 1$, as follows:

$$\frac{1}{r} l_k = \begin{cases} \frac{1}{r} l_1 + \sum_{j=1}^{k-1} l_j & 1 < k \leq s \\ \sum_{j=k-s}^{k-1} l_j & k > s. \end{cases} \quad (7)$$

Figure 5 compares the required server bandwidth as a function of start-up delay for Optimized PB protocols with $n = 2$ and as a function of average and maximum start-up delay for the skyscraper system which assumes $n = 2$. Maximum start-up delay is twice the average start-up delay in skyscraper systems. Note that the performance of the Optimized PB systems is competitive with the skyscraper system, even though each segment is completely received before the beginning of the segment is played. Note also that the performance of the Optimized PB systems improves for lower r.

The Optimized PB segment sizes are also easily generalized for non-negligible latency to join an on-going multicast. In this case, a conservative estimate of the latency to join the multicast transmission is added to the left-hand side of equation (7) and to the time to download the first segment on the right-hand side of the equation for $k \leq s$.

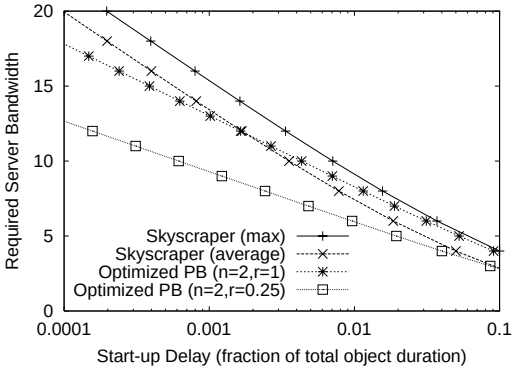


Figure 5: Performance Comparison of Optimized PB and Skyscraper Broadcasts

Required client buffer space for an Optimized PB protocol can be derived by summing the amounts by which the buffer fills during the download time of the initial segment, and during each segment play time for which the download rate exceeds the play rate. Expressed as a fraction of the total object size, the required client buffer space is given by

$$\Big(\min(K,s)l_1 \;+\; (n-1)\sum_{k=1}^{K-s} l_k \;+\; \sum_{k=\max(1,K-s+1)}^{K-s+\lfloor\frac{n-1}{r}\rfloor} ((K-k)r-1)l_k\Big)/\sum_{k=1}^{K} l_k. \tag{8}$$

4.2 Basic Reliable Periodic Broadcast

We next consider modifying the Optimized PB protocols to enable recovery from packet losses in the transmission path to each client. This section generalizes the protocols for the (idealized) case that the cumulative packet loss at the end of receiving each segment is never greater than a tunable parameter p. Sections 4.3 and 4.4 will provide a further optimization and generalizations for environments that have packet loss greater than the assumed upper bound.

A key insight, supported by Figure 4, is that the Optimized PB protocol can be extended to enable recovery from packet loss if *each segment* is delivered by an approximation of a digital fountain. That is, if each segment is "stretched" by an appropriate factor (e.g., 2 or 3) using erasure codes, and if successive transmissions of the segment cycle through the resulting encoded packets, a client can listen to each channel until it has correctly received the number of packets required to reconstruct the respective segment. Assuming perfect decode efficiency[3] and maximum packet loss equal to p, the client will receive the requisite number of packets for segment k by time $1/(1-p)\times l_k/r$. Letting $a = 1/(1-p)$ and assuming segment decode time is negligible, the maximum segment size progression is as follows:

$$a\times\frac{l_k}{r} = \begin{cases} a\times\frac{l_1}{r} + \sum_{j=1}^{k-1} l_j & 1 < k \le s \\ \sum_{j=k-s}^{k-1} l_j & k > s. \end{cases} \tag{9}$$

For a given number of server streams, the (deterministic) start-up delay is equal to $\frac{aT}{r\sum l_k}$. The above segment sizes can be modified to account for non-negligible segment decode times by adding the decode time for segment k to the left-hand side of equation (9) and by adding the decode time for segment 1 to the download time for segment 1 on the right-hand side of the equation. Imperfect decode efficiency can be accounted for by letting a equal the actual decode efficiency times $1/(1-p)$. For example if $p = 0.2$ and decode efficiency is 1.05, $a = 1.05\times 1.25 = 1.3125$. Segment sizes may be capped at some maximum value so as to control client buffer size and decoding time requirements, although in the following performance results we assume all segment sizes are as given by equation (9).

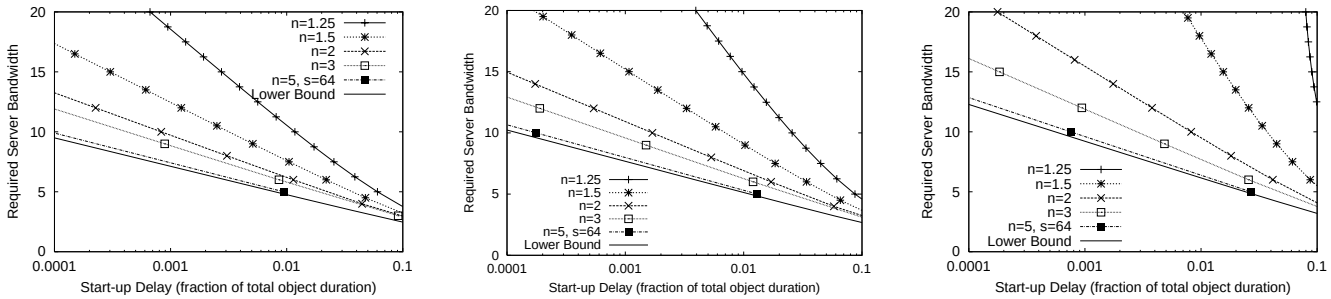
Figures 6 & 7 provide numerical results for the required server bandwidth (in units of the object playback bit rate) for delivery of a single object, as a function of the client start-up delay. Decode efficiency is assumed to be 1.0 and decode time is assumed to be negligible in these figures; required server bandwidth will increase for larger values of either of these parameters. Figure 6 shows the impact of n and p on the required server bandwidth as a function of the start-up delay. As shown in the figure, if maximum packet loss per segment (p) is 10% or less, start-up delay equal to 1% of T is feasible even for aggregate transmission rate to each client (n) is only 1.25 times the media play rate. Moreover, as n and s increase, lower target start-up delays become more feasible (even for higher values of p), and the server bandwidth required by the Optimized PB protocol approaches the lower bound on the required server bandwidth for recovering from loss rate equal to p, which was given in equation (4). Recall that the lower bound assumes unlimited aggregate transmission rate to each client.

In Figure 7 the maximum aggregate transmission rate to a client is fixed (i.e., $n = 2$). Thus, as the maximum number of streams a client listens to (s) increases, (r) decreases and performance improves. However, the figure also shows that decreasing r yields diminishing returns. For example, the benefit obtained for $r < 0.25$ may not be worth the cost of the additional multicast streams. On the other hand, decreasing r yields shorter segments which imply reduced decoding time per segment. All of these factors need to be considered when selecting the value of r for a given implementation of the protocol.

4.3 RPB Protocols for Bursty Packet Loss

In the Basic RPB protocols, each client that observes cumulative packet loss rate less than or equal to p at the end of receiving each segment will be able to recover from packet loss and play the media object without interruption. Furthermore, if a given client observes cumulative packet loss rate less than p at the end of receiving a given segment, that segment can be reconstructed prior to its playout point. In this case, the client can begin listening to later segments earlier than anticipated. This "work-ahead" allows the client

[3] *Decode efficiency* is defined as the ratio of the number of packets that must be received in order to reconstruct a segment to the number of packets in the reconstructed segment. Initially we assume the decode efficiency equal to 1.0 and the time to decode an erasure-coded segment is negligible. These assumptions simplify the derivation of the segment size progressions, and then are easily relaxed.



(a) 3% Packet Loss **(b) 10% Packet Loss** **(c) 25% Packet Loss**

Figure 6: Performance of the Basic RPB Protocols ($a = \frac{1}{1-p}$, default $s = 8$)

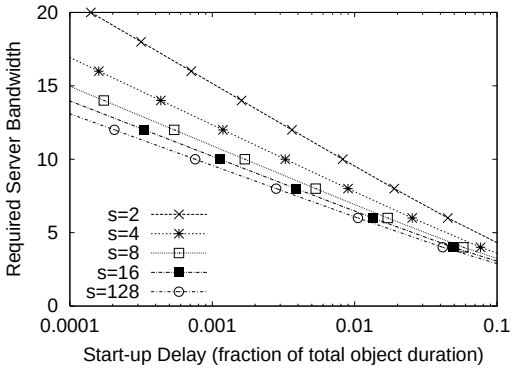


Figure 7: Impact of Streaming Rate on Basic RPB Performance ($n = 2$, 10% Packet Loss)

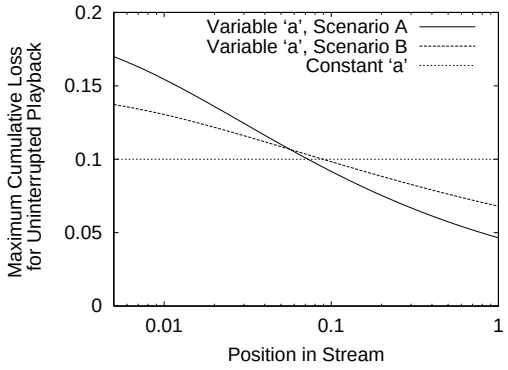


Figure 8: Loss Tolerance Using Variable a ($K = 10, r = 1, n = 2, d = 0.008T$)

to tolerate a higher loss rate than p for a later segment, with no interruption in playback if the cumulative loss probability at the end of receiving the later segment is still less than or equal to p. Permitting clients to begin reception of each segment at an arbitrary point in time enables this useful work-ahead capability.

As discussed in extensive previous work (e.g., [19] and the references therein), packet loss is typically quite bursty. Spikes in the packet loss rate imply that earlier segments in the Basic RPB system require a higher level of loss protection than later segments.

A greater level of protection for earlier segments can be accomplished by letting each segment have a different value a_k that determines the default time the client will listen to the stream for segment k. For a given average loss rate p and $a = 1/(1-p)$ times the decode efficiency, earlier segments will have a_k larger than a and later segments will have a_k lower than a. The RPB segment sizes for the specified values of a_k are easily derived as follows:

$$a_k \times \frac{l_k}{r} = \begin{cases} a_1 \times \frac{l_1}{r} + \sum_{j=1}^{k-1} l_j & 1 < k \leq s \\ \sum_{j=k-s}^{k-1} l_j & k > s. \end{cases} \quad (10)$$

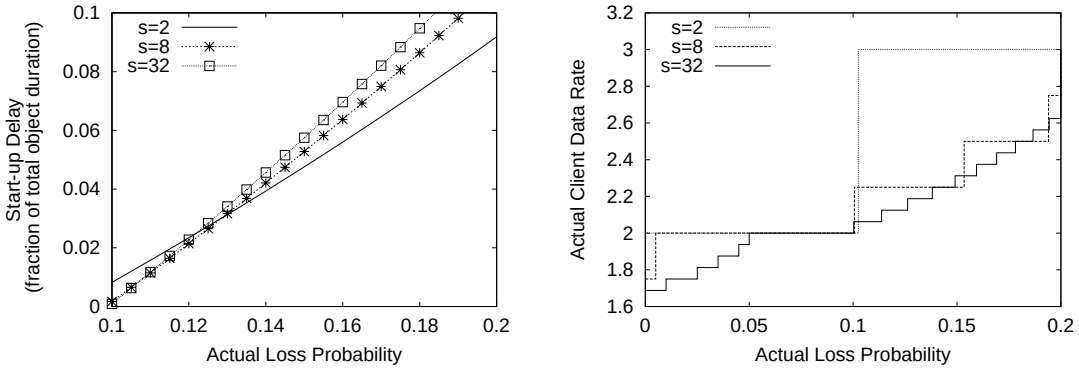
The above specification for RBP systems can be modified to include non-negligible multicast join or segment decode times, as discussed for the Basic RPB protocols.

For a given environment with specified values of n, r, and a_k, the required server bandwidth ($B = K \times r$) can be plotted as a function of start-up delay, to support server provisioning decisions.

Figure 8 shows that a variety of skews in the loss protection can be achieved with a fixed server bandwidth and start-up delay. That is, for $K = 10$, $r = 1$, $n = 2$, and $d = 0.008T$, the figure shows the cumulative loss protection provided with a_k equal for all segments, and the loss protection provided in two example scenarios with variable a_k. For other scenarios, a change in server bandwidth and/or start-up delay may be needed to provide the desired protection for both the initial segments as well as the later segments. In any case, a careful analysis of the traffic characteristics in the implementation environment is needed to properly tailor the loss protection. Note also that in many environments (including the Internet and wireless networks) only a small to moderate skew in the protection may be desired, because more significant spikes in the loss rate indicate that the total transmission rate for the media object should be reduced in order to alleviate congestion in the transmission network. This issue is discussed next.

4.4 Client Heterogeneity

The RPB family of protocols has so far been defined assuming that all clients have (1) the assumed maximum aggregate transmission rate equal to n times the media playback rate, and (2) the assumed maximum cumulative loss probabilities at the end of receiving each segment, which



(a) Start-up Delay vs. Loss Probability (b) Client Data Rate vs. Loss Probability

Figure 9: RPB Performance for Heterogeneous Clients ($B = 10, n = 2, p = 0.1$)

implies that all clients also have (approximately) the same start up delay.

A client with a significantly higher (or lower) aggregate transmission rate should receive a higher (or lower) quality version of the media object, which may be done using one of two approaches. First, several different versions of the object that are encoded for different bit rates might each be delivered using the RPB protocol. If the client may dynamically switch between the different versions based on changes in the transmission channel, then the RPB protocol for each version should have the same parameter values (i.e., n, r, K, and a_k) to minimize interruption in playback when version changes are made. Second, the media object might be compressed using a layered encoding scheme, and the RPB protocol might be applied to each encoded layer. In the simplest case, the parameters of the RPB protocol would be the same for each layer. More complex schemes that allow each layer to have different parameters, for example to implement greater loss protection for the base layer, can also be designed such that the time to receive each segment k is the same for each layer. In any case, a clients with a given aggregate transmission bandwidth (perhaps experimentally determined) subscribes to the appropriate number of layers, possibly dynamically in response to observed changes in the transmission channel.

In some cases, smaller heterogeneities in client capabilities can be accommodated using available tradeoffs among start-up delay, client data rate, and packet loss rate. As an example, Figure 9(a) shows how a client with a higher packet loss rate than specified in the RPB protocol design (i.e., $p = 0.1$) can achieve full packet loss recovery and uninterrupted playback by adopting a higher start-up delay. One possible technique by which this tradeoff could be exploited would involve the client monitoring the packet loss rate prior to playing the first segment (i.e., while receiving the first s segments), and simply delaying beginning of play out if the loss rate is deemed excessive.

As another example, Figure 9(b) shows that if a client has higher aggregate transmission rate than specified in the design of the RPB protocol, the client can tolerate a higher loss rate than specified in the protocol design while still achieving uninterrupted playback, by listening to more streams concurrently than specified in the RPB protocol. The figure illustrates this for three RPB protocols, each designed for $B = 10$, $n = 2$, and $p = 0.1$. Each protocol is designed for a different value of s (i.e., a different segment streaming rate, $r = n/s$). If the actual loss probability is greater than 0.1, the client can achieve uninterrupted playback if the client can achieve the actual transmission rate (greater than 2) given in the curve for the RPB protocol employed to deliver the object. Each step in the curve indicates that the client must listen to one additional stream concurrently to achieve uninterrupted playback. Also note that a slightly larger start-up delay will also be required to ensure that the initial s segments are received in time for playback if these segments have higher than the anticipated loss probability. Conversely, a client that observes lower than anticipated loss probability can in some cases listen to fewer than s streams for uninterrupted playback, as illustrated in the figure for actual loss probability less than 0.1.

In the context of the mechanisms for accommodating heterogeneous clients, the server can also use mechanisms such as redundant transmission paths to reduce the likelihood that the cumulative loss probability to a given client will exceed the maximum that can be tolerated. If, in spite of such mechanisms, the cumulative loss probability observed by a given client exceeds the loss for which the protocol is designed, the only alternative is to switch to a lower quality media stream (by subscribing to fewer layers or a different version). If that fails, interruption in playback is unavoidable.

Further specification of the mechanisms for tolerating heterogeneous client capabilities in RPB systems is left for future work.

5. RELIABLE BANDWIDTH SKIMMING

The Reliable Bandwidth Skimming (RBS) protocols that are developed in this section have at least two advantages compared to the RPB protocols. First, they automatically reduce server bandwidth when client request rate decreases. Second, they support more general (but not zero delay) "fast forward" and "skip ahead" client interactive requests. However, the RBS protocols are somewhat less efficient than the RPB protocols with respect to the amount of data received by each client, and the RBS protocols may be less able to tolerate bursty packet loss without interruption in playback. Furthermore, servers that use the RBS protocols require client requests in order to initiate a new transmission of the media stream, whereas servers that use the RPB

protocols can use a separate multicast stream to announce the current channels that are delivering periodic broadcasts and the parameters of those broadcasts.

The RBS protocols divide the media stream into segments that have fixed duration as short as possible for the specified loss protection. Each segment is "stretched" using erasure codes. The server transmits $1/(1-p)$ encoded packets for each segment using a rate 1.0 *primary* stream and a rate $p/(1-p)$ *secondary* stream, where p is the specified maximum packet loss rate for each segment. The secondary stream is offset in time by the segment duration to provide some protection against burst losses. The client begins playing the media after the first segment has been received on both streams, and decoded. Assuming the packet loss rate for each segment is not greater than p, each subsequent segment is received and decoded prior to its play point.

The primary stream (and the corresponding "redundant" secondary stream) can be merged with earlier primary (and corresponding secondary) streams in the same way that merging is accomplished in the original bandwidth skimming protocols. For example, if the Partition stream merging method described in Section 2 is used, the primary and secondary streams are each transmitted on their own k channels, with fine grained interleaving of the respective packets on the channels. Clients receive $k+1$ primary and $k+1$ secondary substreams, yielding aggregate client transmission rate $n = (1+1/k)(1+p/(1-p))$, in units of the media playback bit rate. Merging the secondary streams that carry the redundant data is the principal extension to previously proposed schemes for live or scheduled broadcasts, which also involve establishing separate streams of redundant data [9, 30].

Various extensions to RBS for tolerating heterogeneous client capabilities are possible. For example, feedback to the server reporting high loss rate could trigger the server to create one or more new channels in a secondary stream for transmitting more redundant data to provide increased protection against packet loss. Further specification of methods for accommodating heterogeneous clients in RBS systems is left for future work.

Figure 10 illustrates the performance of reliable bandwidth skimming protocols for maximum loss rate equal to 10%, and various values for n, the achievable client data rate.[4] For comparison purposes, the graphs also show the lower bound from equation (3), which assumes an unlimited client data rate. The principal observations from this figure are that the RBS protocols adapt to varying client request rate, and in light of the assumed client data rates, yield performance reasonably close to the lower bound.

We have recently implemented a prototype system to experiment with on-demand streaming protocols. This prototype implements bandwidth skimming with aggregate client transmission rate (n) equal to 2, and is installed in the eTeach system which handles 1500 client requests per day when classes are in session at the University of Wisconsin. Clients can request the videos, pause, resume, fast-forward, rewind, and jump to special markers in the content, at arbitrary points in time. The implementation has demonstrated that bandwidth skimming (1) is simple to implement, particularly for the closest target (CT) variant of the protocol [11], (2) is easily extended to support client interactive requests, and (3) can be designed with almost no client feedback for effecting the stream merging. Extensions to implement the RBS protocol are in progress.

[4]These results are obtained from simulations under Poisson request arrivals, although as reported in [12], qualitatively very similar results are obtained for a heavy-tailed distribution of interrequest times modelled by a Pareto distribution. All results in the figure have 95% confidence intervals that are within 5% of the reported values.

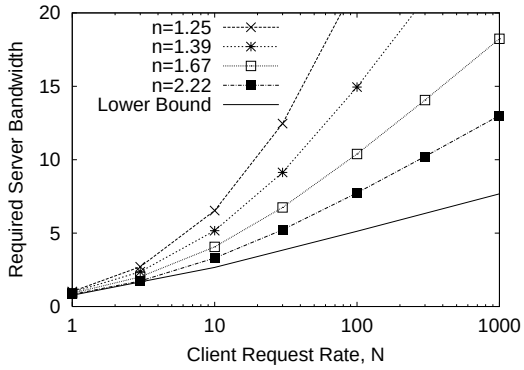


Figure 10: RBS Performance (10% Packet Loss)

6. CONCLUSIONS

This paper has addressed the design of scalable and reliable on-demand delivery of streaming media. New Reliable Periodic Broadcast (RPB) protocols and Reliable Bandwidth Skimming (RBS) protocols were developed and evaluated. The evaluation of the protocols relied in part on simple lower bounds on required server bandwidth for any protocol that provides uninterrupted playback when average packet loss rate is bounded by p. One of the bounds assumes the server provides immediate service to each client; the other assumes the server serves an unlimited number of clients with a specified maximum client start up delay. Each of the new protocols nearly achieves the applicable lower bound, and thus achieves nearly the best possible scalability.

On-going research includes experimental evaluation of the new RPB and RBS protocols, developing congestion control strategies for RPB and RBS systems, developing RPB and RBS protocols for VBR content that has not been fully smoothed, developing RPB systems that have different loss protection for different layers in a layered media stream, and quantifying required server bandwidth for RBS systems that have specified frequencies and types of interactive requests.

7. ACKNOWLEDGMENTS

We would like to thank John Zahorjan for early technical discussions on this topic. We also thank Jussara Almeida, Paul Barford, and the anonymous SIGCOMM 2001 referees for comments that improved the paper presentation.

8. REFERENCES

[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "A Permutation Based Pyramid Broadcasting Scheme for Video On -Demand Systems", *Proc. IEEE ICMCS '96*, Hiroshima, Japan, June 1996.

[2] J. M. Almeida, J. Krueger, D. L. Eager, and M. K. Vernon, "Analysis of Educational Media Server

Workloads", *Proc. NOSSDAV '01*, Port Jefferson, NY, June 2001.

[3] Y. Birk and R. Mondri, "Tailored Transmissions for Efficient Near-Video-On-Demand Service", *Proc. IEEE ICMCS '99*, Florence, Italy, June 1999.

[4] J. C. Bolot, S. Parisis, and D. Towsley, "Adaptive FEC-Based Error Control for Internet Telephony", *Proc. IEEE Infocom '99*, New York, NY, March 1999.

[5] J. Byers, M. Luby, M. Mitzenmacher and A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data", *Proc. ACM Sigcomm '98*, Vancouver, Canada, Sept. 1998.

[6] Y. Cai, K. A. Hua, and K. Vu, "Optimizing Patching Performance", *Proc. MMCN '99*, San Jose, CA, Jan. 1999.

[7] G. Carle and E. W. Biersack, "Survey of Error Recovery Techniques for IP-based Audio-Visual Multicast Applications", *IEEE Network*, Vol. 11, No. 6, Nov./Dec. 1997.

[8] S. W. Carter and D. D. E. Long, "Improving Video-on-Demand Server Efficiency Through Stream Tapping", *Proc. ICCCN '97*, Las Vegas, Sept. 1997.

[9] P. A. Chou, A. E. Mohr, A. Wang, and S. Mehrotra, "FEC and Pseudo-ARQ for Receiver-driven Layered Multicast of Audio and Video", *Proc. IEEE Data Compression Conf.*, Snowbird, UT, March 2000.

[10] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Minimizing Bandwidth Requirements for On-Demand Data Delivery", *IEEE Trans. on Knowledge and Data Engineering*, Sept./Oct. 2001. (Earlier version appears in *Proc. MIS '99*.)

[11] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Optimal and Efficient Merging Schedules for Video-on-Demand Servers", *Proc. ACM Multimedia '99*, Orlando, FL, Nov. 1999.

[12] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Bandwidth Skimming: A Technique for Cost-Effective Video-on-Demand", *Proc. MMCN '00*, San Jose, CA, Jan. 2000.

[13] S. Floyd, V. Jacobson, C. G. Liu, S. McCanne, and L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing", *Proc. ACM Sigcomm '95*, Cambridge, MA, Aug. 1995.

[14] L. Gao, J. Kurose, and D. Towsley, "Efficient Schemes for Broadcasting Popular Videos", *Proc. NOSSDAV '98*, Cambridge, UK, July 1998.

[15] L. Gao and D. Towsley, "Supplying Instantaneous Video-on-Demand Systems Using Controlled Multicast", *Proc. IEEE ICMCS '99*, Florence, Italy, June 1999.

[16] A. Hu, "Video-on-Demand Broadcasting Protocols: A Comprehensive Study", *Proc. IEEE Infocom '01*, Anchorage, AK, April 2001.

[17] K.A. Hua and S. Sheu, "Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems", *Proc. ACM Sigcomm '97*, Cannes, France, Sept. 1997.

[18] K. A. Hua, Y. Cai, and S. Sheu, "Patching: A Multicast Technique for True Video-On-Demand Services", *Proc. ACM Multimedia '98*, Bristol, U.K., Sept. 1998.

[19] W. Jiang and H. Schulzrinne, "Modeling of Packet Loss and Delay and their Effect on Real-Time Multimedia Service Quality", *Proc. NOSSDAV '00*, Chapel Hill, NC, June 2000.

[20] L. Juhn and L. Tseng, "Fast Data Broadcasting and Receiving Scheme for Popular Video Service", *IEEE Trans. on Broadcasting*, Vol. 44, No. 1, March 1998.

[21] X. Li, M. H. Ammar, and S. Paul, "Video Multicast over the Internet", *IEEE Network*, Vol. 13, No. 2, March/April 1999.

[22] X. Li, S. Paul, P. Pancha, and M. Ammar, "Layered Video Multicast with Retransmission (LVMR): Evaluation of Error Recovery Schemes", *Proc. NOSSDAV '97*, St. Louis, MO, May 1997.

[23] J. Nonnenmacher, E. W. Biersack, and D. Towsley, "Parity-Based Loss Recovery for Reliable Multicast Transmission", *IEEE/ACM Trans. on Networking*, Vol. 6, No. 4, Aug. 1998.

[24] J. Nonnenmacher, M. Lacher, M. Jung, G. Carl, and E. W. Biersack, "How Bad is Reliable Multicast Without Local Recovery?", *Proc. IEEE Infocom '98*, San Francisco, CA, April 1998.

[25] J. Paris, S. W. Carter, and D. E. Long, "Efficient Broadcasting Protocols for Video on Demand", *Proc. MASCOTS '98*, Montreal, Canada, July 1998.

[26] C. Perkins, O. Hodson, and V. Hardman, "A Survey of Packet Loss Recovery Techniques for Streaming Audio", *IEEE Network*, Vol. 12, No. 5, Sept./Oct. 1998.

[27] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols", *Computer Communication Review*, Vol. 27, No. 2, April 1997.

[28] L. Rizzo and L. Vicisano, "A Reliable Multicast Data Distribution Protocol Based on Software FEC Techniques", *Proc. HPCS '97*, Greece, June 1997.

[29] S. Sen, L. Gao, J. Rexford, and D. Towsley, "Optimal Patching Schemes for Efficient Multimedia Streaming", *Proc. NOSSDAV '99*, Basking Ridge, NJ, June 1999.

[30] W. Tan and A. Zakhor, "Multicast Transmission of Scalable Video using Receiver-driven Hierarchical FEC", *Packet Video Workshop*, New York, NY, April 1999.

[31] L. Vicisano, L. Rizzo, and J. Crowcroft, "TCP-like Congestion Control for Layered Video Multicast Data Transfer", *Proc. IEEE Infocom '98*, San Francisco, CA, April 1998.

[32] S. Viswanathan and T. Imielinski, "Metropolitan Area Video-on-Demand Service using Pyramid Broadcasting", *Multimedia Systems*, Vol. 4, No. 4, Aug. 1996.

[33] R. Yavatkar, J. Griffoen, and M. Sudan, "A Reliable Dissemination Protocol for Interactive Collaborative Applications", *Proc. ACM Multimedia '95*, San Francisco, CA, Nov. 1995.

[34] X. R. Xu, A. C. Myers, H. Zhang, and R. Yavatkar, "Resilient Multicast Support for Continuous-Media Applications", *Proc. NOSSDAV '97*, St. Louis, MO, May 1997.

APPENDIX

A. MULTICAST RETRANSMISSIONS

This appendix derives a lower bound on the required server bandwidth when retransmissions are multicast, under the assumption that packet losses at each client are independent and occur with probability p. In an actual system there may be some correlation among the packet losses experienced at different clients due to shared links in the transmission paths to these clients. Simple models of plausible correlation structures may be analyzed using a similar approach as described below for independent packet losses.

As in the analysis for the case of no packet loss reviewed in Section 2.3, consider a small portion of the object at some arbitrary time offset x. For an arbitrary client request that arrives at time t, this portion of the object must be delivered no later than time $t+x$. Thus, there is a "sharing window" of duration at most x over which a multicast of this portion may be fruitfully received by new clients.

A lower bound on the required server bandwidth can be obtained by (a) assuming that closely spaced retransmissions experience the same (rather than correlated) loss probability, and (b) neglecting the impact of the time required for retransmissions on the scheduling of "fresh" (i.e., not retransmissions owing to packet loss) multicasts and the size of the sharing window. In the absence of precisely simultaneous client requests, there is only one client for which any particular multicast of the portion at time offset x is "just in time". Under the assumption of uncorrelated loss probabilities, the number of transmissions required to achieve successful delivery to this client is equal to n with probability $p^{(n-1)}(1-p)$, and has average value $\frac{1}{1-p}$. The probability s that some other client that is listening to these transmissions successfully receives the data is given by

$$s = \sum_{n=1}^{\infty} p^{(n-1)}(1-p)(1-p^n) = \frac{1}{1+p}. \tag{11}$$

If one of these clients does *not* successfully receive the data, another "fresh" multicast must be scheduled. The required server bandwidth is minimized if this new multicast is scheduled so as to achieve "just in time" delivery to the earliest such client.

Thus, each fresh multicast of the portion at time offset x incurs on average $\frac{1}{1-p}$ transmissions. Furthermore, not all clients listening to these transmissions may successfully receive the data, and so the minimal average frequency with which these multicasts must be scheduled is increased in comparison to the case of no packet loss. This minimal frequency is identical to the average throughput of a system in which "customers" (representing fresh multicasts and their associated sharing windows) arrive and reside in the system for constant time duration x. Arrivals occur at rate λ when there are no customers present in the system, at rate $\lambda*(1-s)$ when there is one customer present in the system, at rate $\lambda*(1-s)^2$ when there are two customers present in the system, and so on.

The average throughput in this system is identical to that in a similar system, but with exponentially distributed customer residence times of mean duration x. The average throughput can therefore be computed numerically as the solution of an infinite state one-dimensional Markov chain with transition rate from state i to $i+1$ of $\lambda*(1-s)^i$ and from state $i+1$ to i of $\frac{i+1}{x}$, for all $i \geq 0$. A lower bound on required server bandwidth can then be (numerically) obtained by dividing the object into arbitrarily small portions, and summing over all portions, the size of the portion times $\frac{1}{1-p}$ times the throughput computed from the Markov chain analysis for the corresponding time offset x.

B. ASYMPTOTIC ANALYSIS OF RPB

The results presented in Figure 6 suggest that as the parameters n and s of the new periodic broadcast protocols are increased, the start-up delay approaches that of the lower bound. The following asymptotic analysis of the segment size progression supports this result.

Fix n, and let s and K/s grow large, so that each segment becomes infinitesimally small in length and delivery rate, and so that the start-up delay approaches zero. Further, let $l(x)$ denote the length of the $x\frac{s}{n}$'th segment ($0 < x < K\frac{n}{s}$). Equation (9) together with $r = \frac{n}{s}$ yield

$$l(x) = l_{x\frac{s}{n}} = \frac{1}{a} \sum_{j=(x-n)\frac{s}{n}}^{x\frac{s}{n}-1} l_j \frac{n}{s}. \tag{12}$$

Asymptotically the sum can be written as an integral:

$$l(x) = \frac{1}{a} \int_{x-n}^{x} l(y)dy. \tag{13}$$

As can be verified by substitution, the solution to the above equation is

$$l(x) = ce^{bx} \tag{14}$$

where b and c are independent of x, and b is given by

$$b = \frac{1-e^{-bn}}{a}. \tag{15}$$

Thus, asymptotically $l(x) = e^{bn}l(x-n)$, yielding $l_k = e^{bn}l_{k-s}$. Using this equality to substitute for the left-hand side of equation (9), we get that the sum of the lengths of s consecutive segments $js, js+1, ..., (j+1)s-1$ is asymptotically $\frac{a}{r}e^{bn}l_{js}$. Further, the sum of the lengths of the next set of consecutive s segments $(j+1)s, (j+1)s+1, ..., (j+2)s-1$, is greater by a factor of $e^{bn} > 1$. From this geometric progression we can conclude that if the first segment is normalized to length 1, then the sum of the lengths of all K segments with this normalization is asymptotically $\mathcal{O}(e^{bn\frac{K}{s}})$. Thus, since the start-up delay d is equal to a times the length of the first segment, and using $n = sr$, we have that d is $\mathcal{O}(ae^{-brK})$. Solving for the server bandwidth rK yields a required server bandwidth of

$$\frac{1}{b}ln(\frac{1}{d}) + lower\ order\ terms \tag{16}$$

For large n, $\frac{1}{b}$ tends to a. If a is chosen equal to $\frac{1}{1-p}$ (the smallest a for which full recovery from loss can be possible without extra delay at the clients, given loss rate p), we obtain a required server bandwidth that is asymptotically the same as the lower bound given in equation (4).

Interestingly, for $a = 1$ (i.e., the no packet loss case), and general n, we get $b = 1 - e^{-bn}$, implying that $\frac{1}{b}$ is identical to the constant $\eta_{n,\epsilon}$ defined in [10]. This implies that the asymptotic required server bandwidth for general n (and large s), in this no packet loss case, is identical to the conjectured asymptotic lower bound on required server bandwidth for general n from [10].