Scalability of Multicast Delivery for Non-sequential Streaming Access *

Shudong Jin
Computer Science Department
Boston University, Boston, MA 02215
jins@cs.bu.edu

Azer Bestavros
Computer Science Department
Boston University, Boston, MA 02215
best@cs.bu.edu

ABSTRACT

To serve asynchronous requests using multicast, two categories of techniques–stream merging and periodic broadcasting–have been proposed. For sequential streaming access, where requests are uninterrupted from the beginning to the end of an object, these techniques are highly scalable: the required server bandwidth for stream merging grows *logarithmically* as request arrival rate, and the required server bandwidth for periodic broadcasting varies *logarithmically* as the inverse of start-up delay. A sequential access model, however, is inappropriate to model partial requests and client interactivity observed in various streaming access workloads. This paper analytically and experimentally studies the scalability of multicast delivery under a non-sequential access model where requests start at random points in the object. We show that the required server bandwidth for any protocol providing immediate service grows at least as the *square root* of request arrival rate, and the required server bandwidth for any protocol providing delayed service grows *linearly* with the inverse of start-up delay. We also investigate the impact of limited client receiving bandwidth on scalability. We optimize practical protocols which provide immediate service to non-sequential requests. The protocols utilize limited client receiving bandwidth, and they are near-optimal in that the required server bandwidth is very close to its lower bound.

1. INTRODUCTION

Streaming media delivery presents a formidable strain on server and Streaming media delivery presents formidable strain on server and network capacity. With the mushrooming demand for large electronic artifacts stored in Internet servers ranging from Video-on-Demand servers to software repository servers, multicast emerges as a promising scalable delivery technique for such content.

Multicast can be used in both a demand-driven (closed-loop) fashion and a data-centered (open-loop) fashion. In closed-loop approaches, service starts as soon as a request is made. However, as time goes by, it is possible that the service be delegated to an existing multicast stream. For example, consider a scenario in which two clients download a one-hour video, with the second client starting one minute after the first. Service to the second client starts immediately through a dedicated delivery of the first minute of the video to that client, with the remaining fifty nine minutes obtained (and buffered for playout one minute later) by joining the first client's multicast channel. In open-loop approaches, a server multicasts the object or the segments of the object periodically, and clients simply join such multicast channels. Since a server is not interactively responding to request arrivals, clients may have to wait before service could start.

Both closed-loop and open-loop approaches have been well-studied, including the early batching [13, 15], piggybacking [23, 24, 2, 30], and stream tapping/patching [10, 27, 9, 22, 41, 8] techniques, as well as the more recent stream merging [17, 18, 19, 7, 14, 33, 6, 44] and broadcasting protocols [43, 3, 28, 16, 21, 29, 36, 37, 38, 40, 39, 26, 42]. Two particular techniques—stream merging and periodic broadcasting—have been shown to be highly scalable.

Stream merging originated with the work of Eager, Vernon, and Zahorjan [17, 18, 20]. With stream merging, server bandwidth grows *logarithmically* with request arrival rate (or the average number of clients requesting an object simultaneously). Periodic broadcasting was introduced by Viswanathan and Imielinski [43]. With periodic broadcasting, clients may observe a small start-up delay but the required server bandwidth grows *logarithmically* with the inverse of that start-up delay. Both stream merging and periodic broadcasting techniques are built on the assumptions that clients have higher receiving bandwidth than the object playback rate, and that they have local storage to keep prefetched portions of the object temporarily.

The scalability of both stream merging and periodic broadcasting rests on the assumption of sequential access. That is, clients request an object from the beginning and play it without interruption to the end. It is unknown how these techniques scale in a non-sequential access environment, in which clients may request the segments of an object. Indeed, recent studies on the characterization of streaming access workloads [34, 25, 12, 4] have revealed that client access is seldom sequential due to frequent client inter-activity. While several studies have tried to minimize the bandwidth requirement for non-sequential access in Video-on-Demand servers [5, 31, 32, 1, 11, 35], it is still unknown what are the potentials and limitations of multicast delivery in a non-sequential access environment.

We give two example applications with non-sequential access characteristics. The first example is interactive Video-on-Demand for

*This work was supported in part by NSF under Grant ANI-9986397 and ANI-0095988. Shudong Jin was also supported by an IBM Ph.D. Research fellowship.

remote learning in an educational environment, or for press releases in a corporate environment, for example. Here a potentially large number of clients may request an object within a short period of time (*e.g.*, after a lecture is released, or after a press release is put out), but not all of them may settle for a continuous playout from beginning to end. Specifically, clients may jump frequently using VCR functionality such as pause, fast-forward, skip, and rewind. Clearly, it is desirable that the server be able to support a very large number of simultaneous requests while minimizing the start-up delay for these requests. The second example is real-time large software distribution applications. Here a large number of clients may need to download a new software release simultaneously, or within a very short period of time, possibly in reaction to a cyber threat, or to fix a software vulnerability. The entire large software package could be viewed as a streaming object and served using multicast. However, different clients may require different components of the software due to customized installations, for example. This translates to "jumps" in the process of accessing the object.

Paper Contributions and Overview

This paper considers the problem of using multicast to serve non-sequential requests. Under a simple non-sequential streaming access model, we derive a tight lower bound on the required server bandwidth for any protocols providing immediate or delayed service. The lower bound is validated through simulation. It also appears that this lower bound holds for more general cases. Our results indicate that the scalability of multicast delivery under a non-sequential access model is not as good as the logarithmic scalability achievable under a sequential access model. Specifically, we show that for non-sequential access the required server bandwidth for any protocol providing immediate service grows at least as fast as the *square root* of request arrival rate, and that the required server bandwidth for any protocol providing delayed service grows *linearly* with the inverse of the start-up delay. We also study how limited client receiving bandwidth may impact scalability. Finally, we propose practical and very simple delivery protocols that require server bandwidth very close to the lower bound. These protocols provide immediate service to clients, and they assume only limited client receiving bandwidth.

The paper is organized as follows. Section 2 presents some background knowledge and related work on stream merging and periodic broadcasting techniques. In Section 3, we derive the lower bounds on required server bandwidth under a simple non-sequential access model. Section 4 presents simulation results that validate our analytical results under more realistic non-sequential access models. In Section 5, we study the impact of limited client receiving bandwidth. In Section 6, we present optimized multicast delivery protocols for non-sequential access. We conclude in Section 6 with a summary and with directions for future work.

2. BACKGROUND AND RELATED WORK

This section briefly describes two previous techniques, stream merging and periodic broadcasting which utilize multicast delivery to serve streaming media objects. We present results from previous work on the scalability of these techniques when streaming accesses are sequential. In addition, we introduce a non-sequential access model (and notations thereof) used in this paper.

2.1 Stream Merging

In stream merging, a server immediately delivers the object in response to a client's request. This means that the server initiates

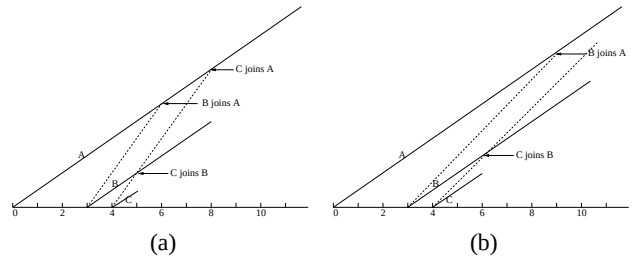


Figure 1: An example of hierarchical stream merging. (a) Client receiving bandwidth is twice the object playback rate. (b) Client receiving bandwidth is 1.5 times the playback rate.

a stream for the client. However, assuming that the client receiving bandwidth is higher than the object's playback rate (it is often assumed that the client can receive up to two streams at the same time), it is possible for the client to listen to a second ongoing stream of the same object, which was initiated by an earlier client. As time goes by, it is possible that the first stream becomes no longer necessary since its future content would have been already prefetched from the second stream. Thus, the client is able to join an ongoing multicast session by virtue of making-up the content it missed from that session using a dedicated stream. This process of merging with multicast sessions that started earlier(and in the process pruning sessions that started later) can be repeated many times, giving rise to *hierarchical* stream merging [20] as opposed to the stream tapping/patching techniques where merging occurs only once for each client.

Figure 1 gives an example where three clients (A, B, and C) request an object at times 0, 3, and 4, respectively. Figure 1(a) assumes that the clients' receiving bandwidth is twice the object playback rate. The server initiates one stream for each client. Client C also listens to the stream for B and prefetches data there. At time 5, the stream initiated for C is no longer necessary since C has already prefetched and will keep prefetching data from the stream for B. From this point on, C starts to listen to the stream for A. Notice that, B starts to listen to the first stream earlier. B virtually joins A at time 6 and C joins A at time 8. Also notice that, after time 6, the stream initiated for B is no longer necessary for B, but C has yet to retrieve segment [3,5] of the object. The server may initiate a stream again for C, or simply prolongs the stream for B by two units of time until C joins A (as shown in the figure).

It is often the case that client receiving bandwidth is less than twice the object playback rate. Bandwidth skimming protocols [19] work well in this case. Each stream is divided into k substreams using fine-grained interleaving, where k is a positive integer. Each substream is then transmitted on a channel with rate equal to $1/k$ times the object playback rate. Stream merging is possible if clients can receive at least $(k+1)$ substreams at the same time. Figure 1(b) gives an example when $k = 2$. In time interval [4,5], C receives two substreams of its own and prefetches only one substream of B. After that, in time interval [5,6], C need only receive one substream of its own and prefetches the two substreams of B. Eventually, C joins B at time 6. Similarly, B joins A at time 9, and C joins A at time 12. A salient feature of bandwidth skimming is that when client receiving bandwidth is slightly higher than the object playback rate (*e.g.*, by 25%), the required server bandwidth is still comparable to that under an unlimited receiving bandwidth

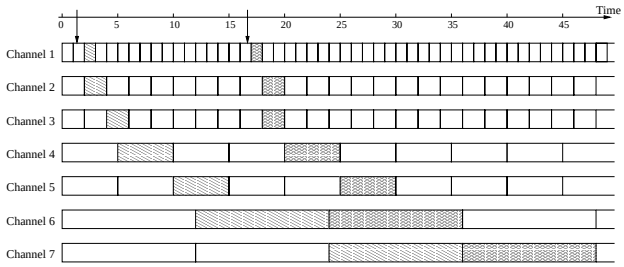


Figure 2: The schedule of Skyscraper broadcasting with 7 segments. The transmission plans for two clients are shown by the shaded segments. Each client receives at most two segments at the same time, and can continuously play out the object after a start-up delay smaller than the duration of the first segment.

assumption.

It has been shown that under both unlimited and limited receiving bandwidth assumptions, the required server bandwidth increases logarithmically with request arrival rates [20]. Thus, stream merging substantially outperforms stream tapping/patching techniques where the required server bandwidth increases as the square root of request arrival rate [22, 20].

2.2 Periodic Broadcasting

In periodic broadcasting schemes, a long object is divided into a series of segments with increasing sizes. Each segment is periodically broadcasted on a dedicated channel. When a client is playing an earlier segment, later segments are prefetched into the client's local buffer. To make this possible, the client must have higher receiving bandwidth than the object playback rate. Like stream merging, it is often assumed that clients can receive two streams/segments at the same time. The segment size progression is made in such a way so that once the client starts playing the first segment, the whole object can be played out continuously.

Two important performance metrics of periodic broadcasting protocols are the required server bandwidth and the start-up delay. The required server bandwidth is proportional to the number of segments, which is fixed and independent of the request arrival rate. The maximum start-up delay is equal to the duration of the first segment. A desirable property of periodic broadcasting protocols is that the small first segment permits a small start-up delay while the larger later segments allow the total number of segments to remain small. To achieve the best tradeoffs between these two metrics, a broadcasting protocol needs to find the quickest segment size progression.

Various periodic broadcasting protocols have been proposed in the literature. To understand the general idea behind these protocols, it suffices to describe one example, the *Skyscraper broadcasting* [28] protocol. Skyscraper broadcasting assumes that client receiving bandwidth is twice the object playback rate. The series of segment sizes is {1,2,2,5,5,12,12,25,25,52,52,...}. The broadcast schedule with seven segments is shown in Figure 2. The figure shows that two clients, starting in time interval (1,2) and (16,17), respectively, are served with delay less than one unit of time, and henceforth can continuously play out the object. These two clients have different transmission schedules, as shown by the shaded segments; none of them needs to receive more than two segments at any time.

The segment size progression of Skyscraper broadcasting has the following property: the size is at least doubled every two steps. Different broadcasting protocols may have different segment size progressions, *e.g.*, geometric series and Fibonacci series, but it is common that the size increases *exponentially*. Thus, the total number of segments (which is proportional to the required server bandwidth) is a logarithmic function of the inverse of the first segment size (which is proportional to the start-up delay).

2.3 Notation and Assumptions

Some of the notation used in this paper are listed in Table 1. Let T be the length of an object in bytes. Assuming that such an object is played out at a constant bit-rate of one byte per unit time, its duration is also T units of time. We consider the following simple model for non-sequential accesses. Request arrivals follow a Poisson process with arrival rate λ. Each request is for a segment of size S which starts from a random point in the object. We assume $S < T$. For simplicity, we assume that the object is cyclic, which means that access may proceed past the end of the object by cycling to the beginning of the object. Parameters T, λ, and S uniquely specify a workload.

For ease of presentation, we introduce two other quantities N and M, which are derived from the three basic parameters T, λ, and S. Let N denote the average number of clients being serviced at the same time. Using Little's Law, it follows that $N = \lambda S$. To assess the scalability of a multicast delivery protocol, it is customary to find out how the server bandwidth B grows as a function of N. Similarly, let M denote the average number of requests over a period of time T. Again, it follows that $M = \lambda T$. Notice that by definition, $M > N$. It also helps to understand that if $S = T/4$ for example, then $M = 4N$.

Let d denote the maximum delay before service for a request is started. Under an immediate service assumption, $d = 0$. Thus, immediate service is a special case of delayed service. For clarity, in our analysis of the next section, we first consider immediate service, and then consider the general delayed service.

Let n denote the client receiving bandwidth in units of object playback rate. For example, $n = 2$ means the client can receive two streams at unit playback rate simultaneously. We first assume that n is unlimited in the derivation of the lower bound on the required server bandwidth. In Section 5, we use simulations to show that when client receiving bandwidth is limited, the required server bandwidth increases slightly.

Finally, considering that storage is not expensive, we assume that the clients have large enough buffers, and thus can always keep prefetched data in their buffers.

2.4 Scalability of Multicast Delivery for Sequential Access

For sequential access, requests start from the beginning of the object and continue uninterruptedly until the end. Thus, $S = T$. Eager $et\ al.$ [20, 33] derived a tight lower bound on the required server bandwidth for any protocol (including stream merging) that provides immediate service. The lower bound is:

$$\begin{aligned} B_{minimum}^{immediate,sequential} &= \int_0^T \frac{dx}{x + 1/\lambda} \\ &= \ln(N+1). \end{aligned} \tag{1}$$

Notation	Definition
T	object duration (or length in bytes, assuming one byte per unit time)
λ	client request arrival rate
S	duration of each request
N	average number of clients being serviced at same time. This quantity is equal to λS.
M	average number of requests arrived in T. This quantity is equal to λT
B	required server bandwidth (in units of object playback rate)
d	maximum start-up delay for each request
n	client receiving bandwidth (in units of object playback rate)

Table 1: Notations used throughout this paper.

The bound is derived by considering an arbitrarily small portion of the object at offset x. This portion is multicasted. Later requests may join the multicast, until the first request after time x which has missed this portion. On average, the server needs to multicast this portion again after time $x + 1/\lambda$. This bound can be extended by adding a start-up delay d as follows:

$$\begin{aligned} B_{minimum}^{delay,sequential} &= \int_0^T \frac{dx}{x + d + 1/\lambda} \\ &= \ln(\frac{N}{\lambda d + 1} + 1). \end{aligned} \quad (2)$$

For periodic broadcast protocols which assume arbitrary large λ, the above lower bound is:

$$B_{minimum}^{periodic,sequential} = \ln(\frac{T}{d} + 1). \quad (3)$$

In summary, with sequential access, (1) the lower bound on the required server bandwidth for any protocol providing immediate service grows logarithmically with request arrival rate, and (2) the lower bound on the required server bandwidth for any protocol providing delayed service grows logarithmically with the inverse of the start-up delay. These results provide the basic scalability arguments of multicast delivery under a sequential access model.

3. MULTICAST DELIVERY WITH NON-SEQUENTIAL ACCESS

In this section, we consider non-sequential access and derive tight lower bounds on the required server bandwidth for any protocol providing immediate service and delayed service. We assume that client receiving bandwidth is unlimited.

3.1 Scalability of Immediate Service Protocols

We consider protocols that provide immediate service under the simple non-sequential access model described in the last section. Let us consider an arbitrarily small portion of the object, say a byte. Assume that at time 0, this byte is multicasted. The question is, when does it need to be multicasted again?

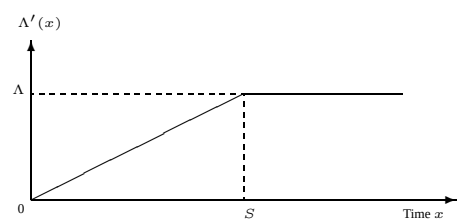
Consider the random variable τ which is the time elapsed until this byte *must* be multicasted again—*i.e.*, τ is the latest point in time beyond which a request would be delayed if the byte were not multicasted again. Clearly, at time τ this byte must be immediately needed by some request because, otherwise, the multicasting of the byte could have waited, which by definition of τ is not the case. Such a request must have been initiated after time 0 because, otherwise, the byte could have been retrieved from the multicast at time 0. Our ultimate goal is to compute how frequently the byte is served—*i.e.*, the expectation of τ. To do so, it suffices to derive the probability density function of τ, denoted $f(.)$. We do so below.

Consider the arrival process $\{X\}$, where event X is the playout of the byte by some request. It is obvious that this arrival process is a Poisson process with an arrival rate $\Lambda = \lambda S/T = N/T$. That is, on average the byte is played out N times in T. As we explained earlier, the playout of the byte by some request at a given moment does not necessarily mean a multicast of the byte at that precise moment. Thus, what we are interested in is another arrival process—namely those arrivals in $\{X\}$ that necessitate that the byte be *multicasted* again. Let $\{X'\}$ denote such an arrival process and let Λ' denote the arrival rate for this process.

Arrivals in $\{X'\}$ are clearly a subset of arrivals in $\{X\}$. Specifically, an arrival in $\{X\}$ that could have retrieved the byte at time 0 must be excluded from $\{X'\}$ since such an arrival would not require a re-multicasting of the byte at time τ. Assume that an arbitrary X occurs at time x. If $x \leq S$, then with probability x/S it is also an event of $\{X'\}$. If $x > S$, then X is certainly (with probability 1) an event of $\{X'\}$. Therefore, the arrival rate Λ' is a function of the arrival time x:

$$\Lambda'(x) = \begin{cases} \Lambda x/S, & \text{if } x \leq S \\ \Lambda, & \text{if } x > S \end{cases}$$

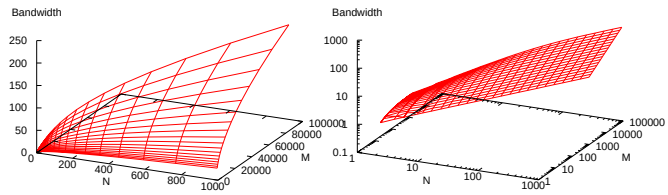
This observation is made more clear in the illustration below.



When $x = 0$, $\Lambda'(x) = 0$, since certainly an event of $\{X\}$ is not that of $\{X'\}$. When x increases, $\Lambda'(x)$ increases linearly until $x = S$. When $x \geq S$, an event of $\{X\}$ is certainly an event of $\{X'\}$, so $\Lambda'(x) = \Lambda$.

We are now ready to derive the marginal and density functions of the random variable τ. To do so, we first compute the expected number of events X' before time x.

$$\begin{aligned} \alpha(x) &= \int_0^x \Lambda'(x)dx \\ &= \begin{cases} \frac{\Lambda x^2}{2S}, & \text{if } x \leq S \\ \Lambda(x - \frac{S}{2}), & \text{if } x > S \end{cases} \end{aligned}$$



(a) Linear scale (b) Log scale

Figure 3: Lower bound on required server bandwidth for immediate service protocols, varying with the number of simultaneous requests N and segment request arrival rate M.

Notice that the probability that there is no arrival over time interval $(0, x)$ is equal to $e^{-\alpha(x)}$ since the arrivals are independent. Hence, the marginal distribution function of τ is:

$$\begin{aligned} F(x) &= P\{\tau > x\} \\ &= 1 - P\{\text{no arrival in interval } (0, x)\} \\ &= \begin{cases} 1 - e^{-\frac{\Lambda x^2}{2S}}, & \text{if } x \le S \\ 1 - e^{-\Lambda(x - \frac{S}{2})}, & \text{if } x > S \end{cases} \end{aligned}$$

Consequently, the probability density function is:

$$\begin{aligned} f(x) &= \frac{dF(x)}{dx} \\ &= \begin{cases} \frac{\Lambda x}{S} e^{-\frac{\Lambda x^2}{2S}}, & \text{if } x \le S \\ \Lambda e^{-\Lambda(x - \frac{S}{2})}, & \text{if } x > S \end{cases} \end{aligned}$$

The remainder of the derivation is straightforward. We compute the expectation of τ as follows:

$$\begin{aligned} E[\tau] &= \int_0^\infty x f(x) dx \\ &= \sqrt{\frac{2S}{\Lambda}} \gamma(\frac{3}{2}, \frac{\Lambda S}{2}) + e^{\frac{-\Lambda S}{2}} (S + \frac{1}{\Lambda}), \end{aligned}$$

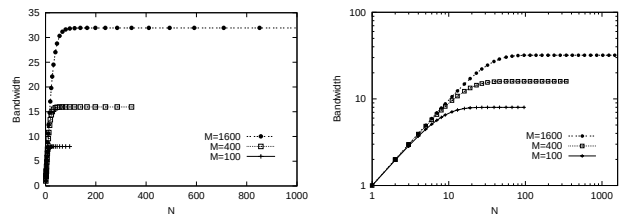
where the incomplete gamma function $\gamma(a, b) = \int_0^b x^{a-1} e^{-x} dx$. Finally, we obtain the average required server bandwidth as

$$\begin{aligned} B &= \frac{T}{E[\tau]} \\ &= \frac{T}{\sqrt{\frac{2S}{\Lambda}} \gamma(\frac{3}{2}, \frac{\Lambda S}{2}) + e^{\frac{-\Lambda S}{2}} (S + \frac{1}{\Lambda})}. \end{aligned}$$

Substituting Λ with N/T and S with NT/M, we obtain the follows

$$B = \frac{1}{\sqrt{\frac{2}{M}} \gamma(\frac{3}{2}, \frac{N^2}{2M}) + e^{\frac{-N^2}{2M}} (\frac{N}{M} + \frac{1}{N})}. \quad (4)$$

The above, seemingly complex result can be greatly simplified. When $N^2 \gg 2M$, the incomplete gamma function $\gamma(\frac{3}{2}, \frac{N^2}{2M})$ approaches the complete gamma function $\gamma(\frac{3}{2}, \infty) = \sqrt{\pi}/2$ very fast. Also, since $e^{\frac{-N^2}{2M}}$ approaches zero very fast, $e^{\frac{-N^2}{2M}} (\frac{N}{M} + \frac{1}{N})$ is insignificant. Hence, $B \approx \sqrt{2M/\pi} \approx 0.797\sqrt{M}$. This means that when M is of the same order as N, the required server bandwidth is at the order of $\sqrt{N}$. This lower bound is much higher than the logarithmic bandwidth requirement of stream merging for sequential requests.



(a) Linear-scale (b) Log-scale

Figure 4: Lower bound varying with N when M is fixed.

Generally, when M increases, B also increases. For example, when $2M = N^2$, we compute the lower bound to be $B \approx 0.54N$, which is comparable to the required server bandwidth under a unicast service. Further increasing M results in diminishing the advantage of multicast delivery, especially when multicast overhead is taken into consideration.

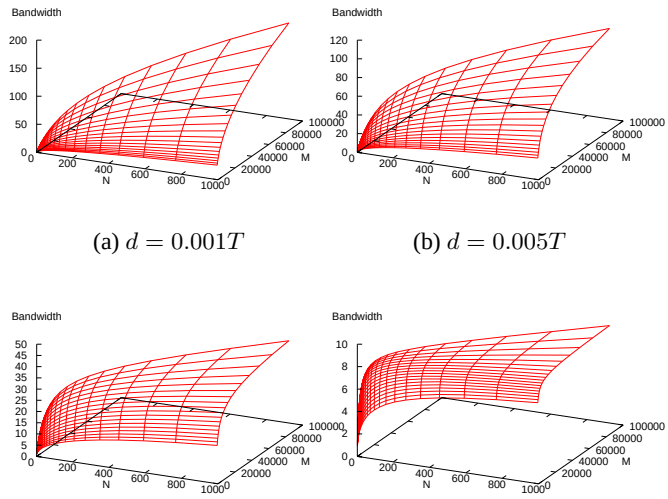
For the general case, it is easy to numerically solve equation (4). We have done so by varying N from 1 to 1000 and varying M from N to $100N$. The results are shown in Figure 3. From Figure 3(a), we observe that the required server bandwidth increases very fast when M increases (S decreases since $S = NT/M$). Eventually, it is close to that of a unicast service. Notice that for sequential access, stream merging techniques have the required server bandwidth lower bound $\log(N + 1)$, which would be at the bottom of the plot (not shown here). Figure 3(b) shows the same plot except that the axes are in log-scale. We observe that the log-value of the lower bound is approximately linear to those of N and M. This is because of the power-law relationship between B and N: the lower bound on bandwidth is at the order of $\sqrt{N}$ when M is close to N, and when M increases to the order of N^2, the lower bound increases to the order of N.

In Figure 4, to better illustrate the lower bound's behavior, we plotted it by fixing M at several values, but varying N up to M. Note that here λ is fixed, so increasing N means increasing S. As can be observed from the figure, when N is small, the required server bandwidth increases linearly with N. When N is large, the required server bandwidth is bounded by the constant $0.797\sqrt{M}$.

Note: The analysis we presented in this section assumes constant segment sizes. For the general case, when the segment size is variable, we found it difficult to derive the lower bound analytically. More details are given in Appendix.

3.2 Scalability of Delayed Service Protocols

We now focus on the more general case—protocols that provide service within a fixed delay. We use the non-sequential access model described in the last section. Requests are satisfied in a slightly different way. Each client tolerates waiting for an interval of d time units (bytes) before it plays the requested segment of the object. During this interval, the client joins other streams to retrieve the bytes that will be played. Once the client starts to play the object, it continues retrieving later bytes whenever available. The client initiates a stream for those bytes that can not be retrieved from other ongoing streams. It is obvious that the imme-



(a) $d = 0.001T$ (b) $d = 0.005T$

(c) $d = 0.02T$ (d) $d = 0.1T$

Figure 5: Lower bound on required server bandwidth for delayed service protocols varying with N and M. The delay is set to several typical values.

diate service model is a special case of this delayed service model when $d = 0$.

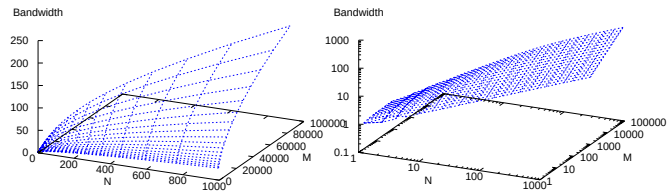
As before, assume that an arbitrary byte is multicasted at time 0. Let τ denote the time when the byte must be multicasted again. Compared to the immediate service case, the expectation of τ increases by d. Hence, we can derive the lower bound of the required server bandwidth to be

$$B = \frac{1}{\sqrt{\frac{2}{M}}\gamma(\frac{3}{2}, \frac{N^2}{2M}) + e^{\frac{-N^2}{2M}}(\frac{N}{M} + \frac{1}{N}) + \frac{d}{T}}. \quad (5)$$

If $d = 0$, this lower bound is consistent with equation (4). When d/T is still very small compared to $\sqrt{1/M}$, this lower bound is close to that of immediate service. Generally, when d increases, this lower bound decreases. However, the rate at which the lower bound decreases is no higher than the inverse of d. This suggests that the use of delayed service is less effective under a non-sequential access model than it is under a sequential access model. Recall that for sequential access, periodic broadcasting techniques reduce bandwidth requirement linearly by increasing service delay logarithmically.

It is straightforward to numerically solve equation (5). We have done so by varying N from 1 to 1000, varying M from N to $100N$, and choosing typical values for d ($0.001T$, $0.005T$, $0.02T$, and $0.1T$). Results are shown in Figure 5. We observe that when d is small, for example $d = 0.001T$, the lower bound on bandwidth is close to that of immediate service shown in Figure 3. When d increases, the lower bound decreases. For all the cases, the lower bound is no larger than T/d.

It is important to notice that, when d is larger, the lower bound approaches $\frac{T}{d}$ faster as N and M increase. Further increases of N and M do not result in higher bandwidth requirement.



(a) Linear scale (b) Log scale

Figure 6: Required server bandwidth for immediate service protocols obtained through simulation, varying with the number of simultaneous requests N and segment request arrival rate M.

4. SIMULATION RESULTS

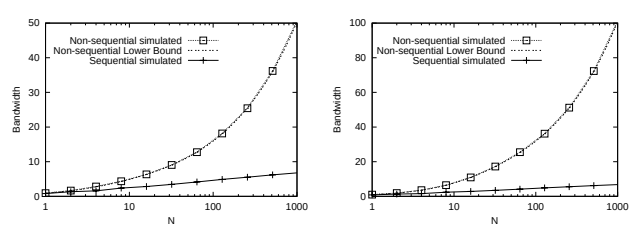
The lower bounds of the previous section were derived using an ideal non-sequential access model assuming independent arrivals and constant segment sizes. To validate these lower bounds and to establish their robustness under more realistic conditions, we performed extensive simulations which we describe in this section.

4.1 Immediate Service

We have written a simulator for a protocol that provides immediate service. Assuming unlimited client receiving bandwidth, a client retrieves later bytes from ongoing streams whenever possible. If a byte cannot be obtained in this way before the time of play, it is multicasted as late as possible (at the time of play) so that other clients can fully utilize it. We varied a number of simulation parameters: N from 1 to 1000 and M from N to $100N$. The results are shown in Figure 6. To obtain each point in the figure, we ran the simulator many times and took the average bandwidth. For clarity, the confidence interval is not shown.

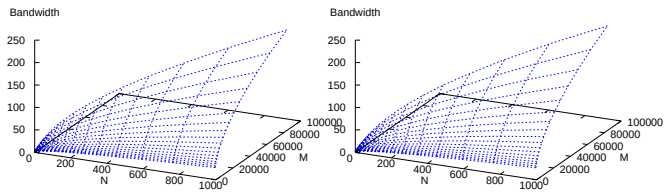
Comparing Figure 6 with Figure 3, we find that in all cases, the average required server bandwidth obtained through simulation is very close to that of our numerical analysis. This is expected since the derived lower bound is *tight*.

Figure 7 shows the average bandwidth for some special cases. The figure shows the required server bandwidth obtained through simulation, the lower bound, and the required server bandwidth (also obtained through simulation) for sequential access for comparison purposes.



(a) $S = T/4$ (b) $S = T/16$

Figure 7: Comparison of the scalability of multicast delivery for immediate service protocols under sequential and non-sequential access models. Non-sequential requests are generated such that each request starts from a random point in the media object.



(a) With uniform distribution. (b) With Pareto distribution.

Figure 8: Required server bandwidth for immediate service protocols obtained through simulation, varying with the number of simultaneous requests N and segment request arrival rate M. Mean segment size is NT/M.

In particular, Figure 7(a) shows the case when $M = 4N$. That is $S = T/4$, *i.e.*, each client randomly requests a segment whose size is equal to one-fourth of the whole object. Figure 7(b) shows the case $M = 16N$. That is $S = T/16$.

From Figure 7 we observe that the required server bandwidth for non-sequential access increases quite fast. Both the results from simulation and the lower bound are much higher than those for sequential access. We have also compared the required server bandwidth for sequential access with its theoretical value (not shown), $\log(N + 1)$, and found that they match well.

For sequential access, the required server bandwidth increases logarithmically with request arrival rate. For non-sequential access, the required server bandwidth increases with the square root of N. In addition, we observed that when $M = 16N$ the required server bandwidth is roughly twice that obtained when $M = 4N$. This is consistent with the fact that the lower bound approximately increases as the square root of M.

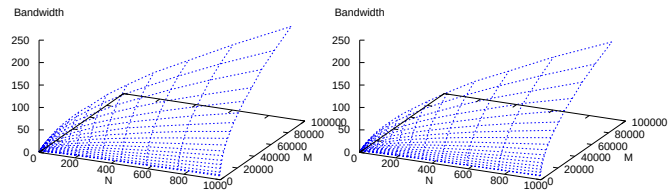
4.1.1 *Effect of Variable Segment Size*

Note that in our analysis (last section) and in the above simulations, we assumed that the segment size is a constant S equal to NT/M. In more realistic workloads, the segment size can vary according to some distribution. Thus, one question is whether the lower bound still holds for various distributions of S.

To answer this question, we generated segment sizes that follow a uniform distribution and a Pareto distribution, respectively. The mean segment size is $S = NT/M$. For the uniform distribution, we let the segment size vary between 0 and $2S$. For the Pareto distribution, we set its shape parameter $\alpha = 2.0$ and computed its scale parameter k to ensure a mean segment size of $S = NT/M$. We have also performed simulations with other values of α (not shown) and found that the corresponding effects on the required server bandwidth were negligible.

Figure 8 shows the results we obtained from simulations with variable segment sizes. Comparing these results with those obtained under a constant segment size assumption (shown in Figure 6), as well as the lower bounds (shown in Figure 3), we found that the differences are almost negligible.

The above results lead us to conclude that with variable segment sizes, the required server bandwidth also increases as the square root of request arrival rate.



(a) All forward jumps. (b) 1/3 of the jumps are backward.

Figure 9: Required server bandwidth for immediate service protocols obtained through simulation, varying with the number of simultaneous requests N and segment request arrival rate M. Requests are generated using an ON-OFF model. ON-segment (request duration) and OFF-segment (jump distance) follow Pareto distribution.

4.1.2 *Effect of Request Correlations*

Our simulations so far were obtained under an assumption that requests are not correlated and that they start anywhere in the object. In the set of simulations we describe next, these assumptions were removed. To do so, we generated requests that exhibit characteristics observed in real streaming access workloads.

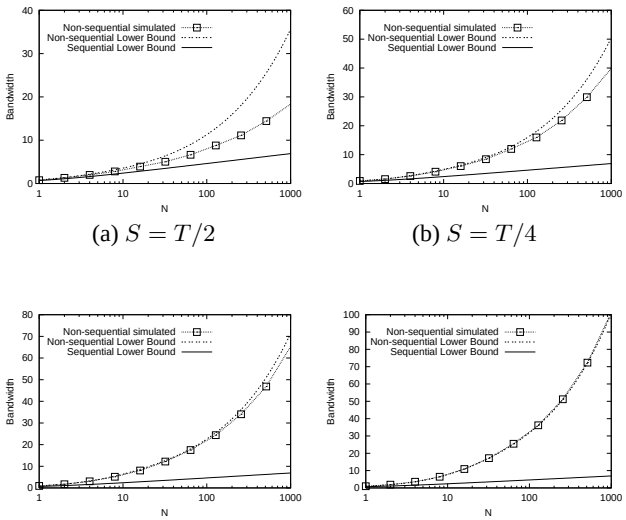
We adopt the following model for client inter-activity. Each client starts a *session* from the beginning of an object. After receiving a segment of the object (the ON-segment), the client skips a portion of the object (the OFF-segment). This process repeats until a request or a jump goes beyond either end of the object.

In prior studies that characterized streaming access workloads [34, 4], it has been observed that the distributions of ON-segments tend to be heavy-tailed. In particular, the Pareto distribution was found to be a close fit. Thus, in our simulations, we generated requests that exhibited such properties. For ON periods, we used a Pareto distribution with parameter $\alpha = 2$ and we set the scale parameter k to achieve an average segment size of $S = NT/M$. For OFF periods, we also used a Pareto distribution with parameter $\alpha = 2$ and we set the scale parameter k to achieve an average jump distance of $S/2$. Results from these simulations are shown in Figure 9.

In other simulation experiments (not shown) we changed the average jump distance, as well as other parameters. Our results suggest that the required server bandwidth appears not to be very sensitive to these variations.

Figure 9(a) shows the results when only forward jumps are allowed. That is, each client requests a segment and skips a segment to continue, and so on until the end of the object. Figure 9(b) shows the results when 33% of the jumps were backward jumps. Comparing these results to the lower bound in Figure 3 and the simulation results in Figure 6 and 8, we found the required server bandwidth to be very close.

Comparing Figure 9(b) to Figure 9(a), we found that with backward jumps, the required server bandwidth decreases slightly. We have estimated the difference and found it to be up to 15%. This can be explained as follows. In our simulation, we assumed that each client has a large enough buffer to keep the segments of the object that have been played. With backward jumps, it is possible that the client plays a portion of the object kept in the buffer. This is equivalent to introducing a start-up delay for the request.



(a) $S = T/2$ (b) $S = T/4$

(c) $S = T/8$ (d) $S = T/16$

Figure 10: Required server bandwidth for immediate service protocols obtained through simulation, varying with the number of simultaneous requests N. M is set to several values. Requests are generated using an ON-OFF model.

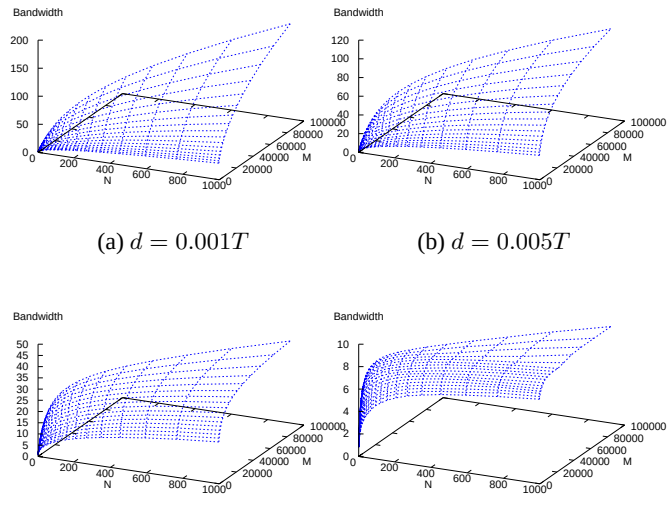
Hence, it reduces the required server bandwidth. Nevertheless, the presence of backward jumps does not change the asymptotic square root lower bound.

From Figure 9, we also observe that when M is small, the required server bandwidth appears lower than those in Figures 3, 6, and 8. This is expected. When M is smaller, the average request duration is closer to T, and since the clients start to retrieve the object from the beginning, requests tend to be "sequential". To understand this more clearly, we zoom in on the plot in Figure 9(a). In Figure 10, we show several special cases when $M = 2N$, $M = 4N$, $M = 8N$, and $M = 16N$. Note that when M is smaller, *e.g.*, $M = 2N$, the average request duration is $T/2$. In this case, access is closer to "sequential", and the required server bandwidth is closer to the lower bound for sequential access. This is evident in Figure 10(a). When M increases, access becomes "less sequential", resulting in a required server bandwidth that is more accurately predicted by the lower bound for non-sequential access. This is evident in Figure 10(b)-(d). To summarize, with only very few jumps during the time of a complete object playout, the required server bandwidth increases at least as the square root of the request arrival rate.

4.2 Delayed Service

We have also used simulations to validate the required server bandwidth of delayed service protocols. We varied simulation parameters N from 1 to 1000 and varied M from N to $100N$. We chose typical values for d—namely $0.001T$, $0.005T$, $0.02T$, and $0.1T$.

In each simulation run, we first generate a sequence of requests. Each request is delayed for time d. During this delay, later bytes are fetched from ongoing streams whenever possible. When a client is playing the object, later bytes can be still retrieved from ongoing streams. Any byte that cannot be obtained in this manner is retrieved from the server directly, and *as late as possible* such that later clients can fully utilize it.



(a) $d = 0.001T$ (b) $d = 0.005T$

(c) $d = 0.02T$ (d) $d = 0.1T$

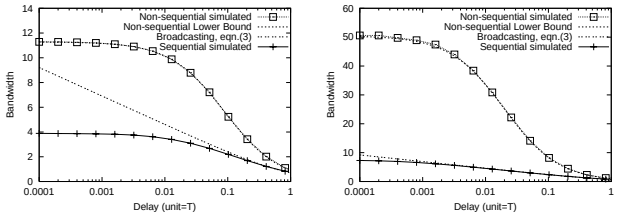
Figure 11: Required server bandwidth obtained through simulation varying with N and M. The delay is set to several typical values.

The results of our simulations are shown in Figure 11. Comparing those results to the ones anticipated in Figure 5, we find that, in all cases, the average required server bandwidth obtained through simulations is close to that we obtained through analysis.

Figure 12 shows how the required server bandwidth varies with delay d. We choose $M = 4N$, *i.e.*, each request retrieves one-fourth of the whole object. Figure 12(a) shows results when $N = 50$, representing a less popular object, and Figure 12(b) shows the case when $N = 1000$, representing a highly popular object. In each case, we plot the lower bound for non-sequential access, as well as that obtained through simulations. In addition, for comparison purposes, we also plot the bandwidth-delay relationship for sequential access. In particular, we plot the minimum required server bandwidth for periodic broadcasting in equation (3), and the required server bandwidth measured from simulation for stream merging (assuming unlimited client receiving bandwidth). Our observations are summarized as follows.

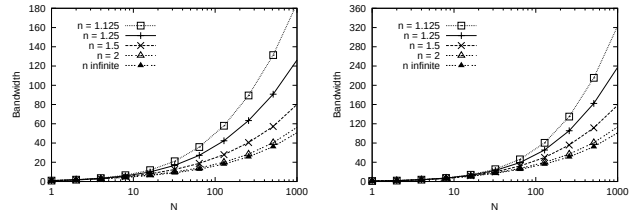
First, the required server bandwidth under a non-sequential access model is much higher than that under a sequential access model. The difference is more pronounced for more popular objects. Notice that when $d \to 0$, the lower bound on required bandwidth under a non-sequential access model is close to $0.797\sqrt{M}$ which is much higher than the $\log(N+1)$ lower bound under a sequential access model.

Second, under a sequential access model, stream merging techniques achieve lower required server bandwidth than broadcasting when delay is small. This is more obvious for less popular objects. This is because stream merging has a lower bound of $\log(N+1)$ when $d \to 0$, whereas broadcasting techniques have a lower bound of $\log(T/d+1)$. So when $d < T/N$, the required server bandwidth for broadcasting is higher. In fact, in this figure, the simulation results for stream merging under a sequential access model are close to the bound given in equation (2).



(a) $N = 50$ (b) $N = 1000$

Figure 12: Required server bandwidth varying with d.



(a) $S = T/4$ (b) $S = T/16$

Figure 13: Required server bandwidth of immediate service protocols when client receiving bandwidth is limited.

5. IMPACT OF LIMITED BANDWIDTH

So far, we assumed that clients have unlimited receiving bandwidth ($n = \infty$). This section uses simulations to study how limited client receiving bandwidth affects the scalability of multicast delivery for immediate service protocols. In order for multicast delivery to be possible, the client receiving bandwidth $n > 1.0$ must hold.

In the context of sequential access, there are advanced techniques for scheduling client requests using multicast streams. Eager $et\ al.$ [18] proposed several *hierarchical multicast stream merging* techniques which progressively merge asynchronous client requests into larger and larger groups. One problem that must be addressed relates to the scheduling of these mergers. Several heuristic policies were proposed. In particular, assuming $n = 2$, the *closest target* policy requires each client to join the most recently initiated earlier stream that is still alive. Simulations showed that this policy performs very close to the off-line optimal algorithm (with known client request arrivals in advance), which is obtained by solving a dynamic programming problem.

Under a non-sequential access model, we modify the closest target merging policy when $n = 2$. The key point is how we define the *closest target* for each client. Since the requests may start anywhere in the object, the closest target is not necessarily the most recently initiated stream. Instead, we define the closest target of each client as the multicast stream that will be played by the client in the nearest future. The intuition behind this choice is that it is critical for the client to prefetch the portion of the data that will soon be played. In addition, the closest target will be redefined when the target itself is merged or is terminated.

When $n < 2$, we modify the *bandwidth skimming* technique [19]. The following is a straightforward adaption of the bandwidth skimming technique described in Section 2.1. Each multicast stream is divided into k substreams, where k is a positive integer. Each substream is then multicasted with rate equal to $1/k$ times the object playback rate. Each client is assumed to be capable of receiving at least $(k + 1)$ substreams simultaneously (thus $n = 1 + 1/k$). The closest target policy is then applied to the substreams: whenever the client has idle bandwidth, it listens to the substreams whose data will be played in the nearest future. The client may listen to up to $(k + 1)$ substreams.

We developed a simulator for this stream merging/bandwidth skimming technique. Figure 13 shows the required server bandwidth we obtained through simulations in which we varied the number of concurrent clients N. The value of n is varied from 1.125 to ∞. In Figure 13(a), we set $M = 4N$, *i.e.*, $S = T/4$, each request is for one-fourth of the object. In Figure 13(b), we set $M = 16N$, *i.e.*, $S = T/16$.

From this figure, we observe that when a client's receiving bandwidth is limited, the required server bandwidth is only slightly higher. For example, when $N = 1000$ and $n = 1.5$ (*i.e.*, one-third of the client bandwidth is used for prefetching), our simulations indicate that the required server bandwidth is about 1.69 times that required under unlimited client receiving bandwidth assumption ($n = \infty$). When $n = 2$, our simulations indicate that the required server bandwidth is very close to that under unlimited client receiving bandwidth assumption. We estimate that the difference is around 12% for large values of N. These results suggest that even with a limited client receiving bandwidth, it is possible for a protocol to have required server bandwidth quite close to its lower bound. In the next section, we study practical multicast delivery protocols that achieve this goal.

6. PRACTICAL MULTICAST DELIVERY

The protocols considered in the previous sections either assume unlimited client receiving bandwidth or are too sophisticated to be practical. For example, in the modified closest target merging policy, each client may join and leave multicast streams frequently. Such a behavior may incur high overhead on servers. In this section, we describe practical multicast delivery protocols and optimize them by minimizing the required server bandwidth.

6.1 Protocols

We consider the following protocol which provides immediate service to the clients: *The server multicasts an object for every interval of length x. Each client joins the temporally closest multicast stream. The missed portion of the requested segment is immediately unicasted from the server.*

The above protocol is simple in that for each request, the client need only join one multicast stream. Also, the above protocol is readily usable by a client whose receiving bandwidth is twice the object playback rate, *i.e.*, $n = 2$. Both the multicast and unicast streams are sent at playback rate. Assuming that a client requests a segment which was most recently multicasted t units of time ago, then the client receives a unicast stream of length t, while concurrently prefetching data from that multicast stream.

Notice that it is straightforward to generalize the above protocol for clients with lower receiving bandwidth, *i.e.*, $1 < n < 2$. To do so, we again capitalize on the ideas from bandwidth skimming techniques. Namely, each stream (both unicast or multicast) is divided into k substreams using fine-grained interleaving, where k is a positive integer. Each substream is sent with rate equal to $1/k$ times the playback rate. The clients can receive $(k+1)$ substreams (thus $n = 1+1/k$). Assume a client requests a segment which was

most recently multicasted t units of time ago. The client receives k unicast substreams immediately from the server and meanwhile it prefetches data from one multicast substream. Then, t units of time later, the client need only receive $(k-1)$ unicast substreams, and can prefetch two multicast substreams. Another t units of time later, the client need only receive $(k-2)$ can prefetch three multicast substreams, and so on. Eventually, no unicast substream is needed.

6.2 Protocol Optimization

To optimize our protocol, we determine the value of x, which controls the frequency of multicasting the object. Assume that each request is for a segment of constant length S. We first compute the average cost of unicast. Assume a client requests a segment which was most recently multicasted t units of time ago. From the description of the protocols above, it is not difficult to find that the total duration of the unicast substreams is $\sum_{i=1}^{k} it = k(k+1)t/2$, assuming $S \geq kt$. On average this is equal to $k(k+1)x/4$. Since the sending rate is $1/k$ times the playback rate, the number of bytes unicasted is $(k+1)x/4$. Over a period of time T, there are M requests, so there are $(k+1)Mx/4$ bytes unicasted on average. In addition, over a period of time T, the object is multicasted T/x times, so there are T^2/x bytes multicasted on average.

We are now ready to optimize our protocol by minimizing the total cost of the protocol over a period of time T. This cost is given by:

$$g(x) = \frac{(k+1)M}{4}x + \frac{T^2}{x}.$$

It is not difficult to find the optimal solution $g^* = \sqrt{(k+1)M}T$ when $x^* = 2T/\sqrt{(k+1)M}$. This yields a required server bandwidth of $\sqrt{(k+1)M}$.

When $k = 1$ and $n = 2$, the required server bandwidth is $1.414\sqrt{M}$, which is approximately 1.773 times the lower bound given in equation (4). This means that such a simple protocol needs less than twice the server bandwidth required for the rather impractical protocols we assumed in our simulations. Even when k increases (*i.e.*, n approaches unity), the required server bandwidth for this simple protocol increases only as the square root of $k+1$. For example, when 33% of the client bandwidth is used for prefetching ($k = 2$ and $n = 1.5$), the required server bandwidth is only $1.732\sqrt{M}$ which is about 2.173 times the lower bound.

Although it is possible to further decrease the required server bandwidth by using more sophisticated protocols, or fine tuning the simple protocol we described here, the payoff from such an exercise is fairly limited. Recall that in the last section, our simulation results have shown that when $n = 1.5$, the sophisticated earliest target first merging algorithm we discussed requires server bandwidth roughly equal to 1.69 times the lower bound—allowing only a limited "room for improvement" over the 2.173 times the lower bound achieved using the protocol described above.

7. CONCLUSION

In this paper, we have analytically derived tight lower bounds on the required server bandwidth for protocols in a multicast environment when access to streaming objects is not sequential. In particular, we have shown that in such systems, the required server bandwidth for any protocol providing immediate service grows as the square root of the request arrival rate, and that the required server bandwidth of any protocol providing delayed service is inversely proportional to the maximum allowable start-up delay. The robustness of our analytical results have been confirmed using extensive simulations under realistic workloads. Also, the impact of limited client receiving bandwidth was investigated. Finally, based on our findings, we proposed a practical, near-optimal multicast-based delivery protocol, which results in a server bandwidth requirement that is fairly close to its lower bound under both abundant and limited client receiving bandwidth assumptions.

Our findings suggest that for non-sequential access, multicast delivery is not a panacea for scalability. Therefore, for large-scale content delivery applications that require non-sequential access of large electronic artifacts, for example, interactive video delivery and real-time software distribution, we should seek alternative or complementary techniques that increase the scalability of streaming delivery mechanisms. Such techniques include caching, buffering, and replication, among others.

ACKNOWLEDGMENTS

The authors would like to thank Mark Crovella for providing a few references on streaming access workload characterization. The authors are also grateful to the anonymous reviewers for their insightful and helpful comments.

APPENDIX: Variable Segment Size Analysis

Assume that the segment size is a random variable, following a distribution with a general density function $g(y)$, $a < y < b$. The mean segment size is S. We hope to follow the steps in Section 3 to compute the lower bound on the required server bandwidth.

Let $\{X\}$ and $\{X'\}$ be as defined in Section 3. First we need to compute $\Lambda'(x)$, the arrival rate of $\{X'\}$. Assume an arbitrary X occurs at time x. The probability that this X is also an event of $\{X'\}$ can be computed as

$$\int_a^x \frac{yg(y)}{S}dy + \int_x^b \frac{yg(y)}{S}\frac{x}{y}dy. \tag{6}$$

This expression is explained as follows. Here $\frac{yg(y)}{S}$ is the probability density that the event X occurs as a client requests a segment of length y. Note that it is more likely X occurs as the result of a longer request. If $y < x$, then certainly the event X is also an event of $\{X'\}$. In this case, $\int_a^x \frac{yg(y)}{S}dy$ gives the cumulative probability. If $y \geq x$, then with probability $\frac{x}{y}$, the event X is also an event of $\{X'\}$. In this case, $\int_x^b \frac{yg(y)}{S}\frac{x}{y}dy$ gives the cumulative probability.

It is not difficult to verify that the deterministic model in Section 3 is a special case.

For the general case, equation (6) cannot be solved.

We have considered several special cases with variable segment sizes including: (i) uniform distribution $g(y) = \frac{1}{2S}$, $0 < y < 2S$; and (ii) exponential distribution $g(y) = \frac{1}{S}e^{-y/S}$, $y > 0$. Although we are able to compute equation (6) and the exact expression of $\Lambda'(x)$, when we proceed as in Section 3, the derivation of the lower bound becomes increasingly complex.

REFERENCES

[1] E. L. Abram-Profeta and K. G. Shin. Providing unrestricted VCR functions in multicast video-on-demand servers. In *Proceedings of ICMCS*, June 1998.

[2] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On optimal piggybacking merging policies for video-on-demand systems. In *Proceedings of SIGMETRICS*, May 1996.

[3] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. A permutation-based pyramid broadcasting scheme for video-on-demand systems. In *Proceedings of ICMCS*, 1996.

[4] J. Almeida, J. Krueger, D. Eager, and M. Vernon. Analysis of educational media server workloads. In *Proceedings of NOSSDAV*, June 2001.

[5] K. C. Almeroth and M. H. Ammar. The use of multicast delivery to provide a scalable and interactive video-on-demand service. *IEEE Journal on Selected Areas in Communications*, 14:1110–1122, 1996.

[6] A. Bar-Noy, J. Goshi, R. E. Ladner, and K. Tam. Comparison of stream merging algorithms for media-on-demand. In *Proceedings of MMCN*, January 2002.

[7] A. Bar-Noy and R. E. Ladner. Competitive on-line stream merging algorithms for media-on-demand. In *Proceedings of Symposium on Discrete Algorithms*, January 2001.

[8] Y. Cai and K. A. Hua. An efficient bandwidth-sharing technique for true video on demand systems. In *Proceedings of ACM MULTIMEDIA*, November 1999.

[9] Y. Cai, K. A. Hua, and K. Vu. Optimizing patching performance. In *Proceedings of MMCN*, 1999.

[10] S. W. Carter and D. D. E. Long. Improving video-on-demand server efficiency through stream tapping. In *Proceedings of ICCCN*, September 1997.

[11] S. W. Carter, D. D. E. Long, and J. Paris. An efficient implementation of interactive video-on-demand. In *Proceedings of MASCOTS*, August 2000.

[12] M. Chesire, A. Wolman, G. Voelker, and H. Levy. Measurement and analysis of a streaming workload. In *Proceedings of USITS*, March 2001.

[13] T. Chiueh and C. Lu. A periodic broadcasting approach to video-on-demand service. In *Proceedings of MMCN*, 1995.

[14] E. G. Coffman, J. P. Jelenhovic, and P. Momcilovic. Provably efficient stream merging. In *Proceedings of Web Caching Workshop*, June 2001.

[15] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *ACM Multimedia Systems Journal*, 4(3):112–121, 1996.

[16] D. Eager and M. Vernon. Dynamic skyscraper broadcasts for video-on-demand. In *Proceedings of MIS*, 1998.

[17] D. Eager, M. Vernon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. In *Proceedings of MIS*, 1998.

[18] D. Eager, M. Vernon, and J. Zahorjan. Optimal and efficient merging schedules for video-on-demand servers. In *Proceedings of ACM MULTIMEDIA*, November 1999.

[19] D. Eager, M. Vernon, and J. Zahorjan. Bandwidth skimming: A technique for cost-efficient video-on-demand. In *Proceedings of MMCN*, January 2000.

[20] D. Eager, M. Vernon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. *IEEE Transactions on Data and Knowledge Engineering*, 13, 2001.

[21] L. Gao and D. Towsley. Efficient schemes for broadcasting popular videos. In *Proceedings of NOSSDAV*, June 1998.

[22] L. Gao and D. Towsley. Supplying instantaneous video-on-demand services using controlled multicast. In *Proceedings of ICMCS*, June 1999.

[23] L. Golubchik, J. C. S. Liu, and R. R. Muntz. Reducing I/O demand in video-on-demand storage servers. In *Proceedings of SIGMETRICS*, 1995.

[24] L. Golubchik, J. C. S. Liu, and R. R. Muntz. Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers. *ACM Multimedia Systems Journal*, 4(3):140–155, 1996.

[25] N. Harel, V. Vellanki, A. Chervenak, G. Abowd, and U. Ramachandran. Workload of a media-enhanced classroom server. In *Proceedings of Workshop on Workload Characterization*, 1999.

[26] A. Hu. Video-on-demand broadcasting protocols: A comprehensize study. In *Proceedings of INFOCOM*, April 2001.

[27] K. A. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true video-on-demand services. In *Proceedings of ACM MULTIMEDIA*, 1998.

[28] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *Proceedings of SIGCOMM*, September 1997.

[29] L. Juhn and L. Tseng. Harmonic broadcasting for video-on-demand service. *IEEE Transactions on Broadcasting*, 44(1):100–105, 1998.

[30] S. W. Lau, J. C. S. Liu, and L. Golubchik. Merging video streams in a multimedia storage server: Complexity and heuristics. *ACM Multimedia Systems Journal*, 6(1):29–42, 1998.

[31] V. O. Li, W. Liao, X. Qiu, and E. Wong. Performance model of interactive video-on-demand systems. *IEEE Journal on Selected Areas in Communications*, 14:1099–1109, 1996.

[32] W. Liao and V. O. Li. The split and merge protocol for interactive video-on-demand. *IEEE Multimedia*, 4:51–62, 1997.

[33] A. Mahanti, D. Eager, M. Vernon, and D. Sundaram-Stukel. Scalable on-demand media streaming with packet loss recovery. In *Proceedings of SIGCOMM*, August 2001.

[34] J. Padhye and J. Kurose. An empirical study of client interactions with a continuous-media courseware server. In *Proceedings of NOSSDAV*, June 1998.

[35] J. Paris. An interactive broadcasting protocol for video-on-demand. In *Proceedings of IPCCC*, April 2001.

[36] J. Paris, S. W. Carter, and D. D. E. Long. Efficient broadcasting protocols for video on demand. In *Proceedings of MASCOTS*, July 1998.

[37] J. Paris, S. W. Carter, and D. D. E. Long. A low bandwidth broadcasting protocol for video on demand. In *Proceedings of ICCCN*, 1998.

[38] J. Paris, S. W. Carter, and D. D. E. Long. A hybrid broadcasting protocol for video on demand. In *Proceedings of MMCN*, 1999.

[39] J. Paris, S. W. Carter, and D. D. E. Long. A reactive broadcasting protocol for video on demand. In *Proceedings of MMCN*, January 2000.

[40] J. Paris, D. D. E. Long, and P. E. Mantey. Zero-delay broadcasting protocols for video on demand. In *Proceedings of ACM MULTIMEDIA*, November 1999.

[41] S. Sen, L. Gao, J. Rexford, and D. Towsley. Optimal patching schemes for efficient multimedia streaming. In *Proceedings of NOSSDAV*, June 1999.

[42] S. Sen, L. Gao, and D. Towsley. Frame-based periodic broadcast and fundamental resource tradeoffs. In *Proceedings of IPCCC*, April 2001.

[43] S. Viswanathan and T. Imielinski. Pyramid broadcasting for video on demand service. In *Proceedings of MMCN*, 1995.

[44] Y. Zhao, D. Eager, and M. Vernon. Efficient delivery techniques for variable bit rate multimedia. In *Proceedings of MMCN*, January 2002.