Minimizing Bandwidth Requirements for On-Demand Data Delivery*

Derek Eager
Dept. of Computer Science
Univ. of Saskatchewan
eager@cs.usask.ca

Mary Vernon
Computer Sciences Dept.
Univ. of Wisconsin
vernon@cs.wisc.edu

John Zahorjan
Dept. of Computer Science
Univ. of Washington
zahorjan@cs.washington.edu

Abstract

Two recent techniques for multicast or broadcast delivery of streaming media can provide immediate service to each client request, yet achieve considerable client stream sharing which leads to significant server and network bandwidth savings. This paper considers (1) how well these recently proposed techniques perform relative to each other, and (2) whether there are new practical delivery techniques that can achieve better bandwidth savings than the previous techniques over a wide range of client request rates.

The principal results are as follows. First, the recent partitioned dynamic skyscraper technique is adapted to provide immediate service to each client request more simply and directly than the original dynamic skyscraper method. Second, at moderate to high client request rates, the dynamic skyscraper method has required server bandwidth that is significantly lower than the recent optimized stream tapping/patching/controlled multicast technique. Third, the minimum required server bandwidth for any delivery technique that provides immediate real-time delivery to clients increases *logarithmically* (with constant factor equal to one) as a function of the client request arrival rate. Furthermore, it is (theoretically) possible to achieve very close to the minimum required server bandwidth if client receive bandwidth is equal to two times the data streaming rate and client storage capacity is sufficient for buffering data from shared streams. Finally, we propose a new practical delivery technique, called *hierarchical multicast stream merging (HMSM)*, which has required server bandwidth that is lower than partitioned dynamic skyscraper, and is reasonably close to the minimum achievable required server bandwidth over a wide range of client request rates.

Keywords: streaming media, scalable protocols, multicast, performance evaluation, video-on-demand

1 Introduction

This paper considers the server (disk I/O and network I/O) bandwidth required for on-demand real-time delivery of large data files, such as audio and video files[1]. Delivery of the data might be done via the Internet or via a broadband (e.g., satellite or cable) network, or some combination of these networks.

We focus on popular, widely shared files, such as popular news clips, product advertisements, medical or recreational information, television shows, or successful distance education content, to name a few examples. Due to the large size and the typical skews in file popularity, for the most popular files, one can expect many new requests for the file to arrive during the time it takes to stream the data to a given client.

Prior research has shown that the server and network bandwidth required for on-demand delivery of such files can be greatly reduced through the use of multicast delivery techniques[2]. A simple approach is to make requests wait for service, hoping to accumulate multiple requests in a short time that can then all be served by a single multicast stream [DaSS94]. A second approach (called *piggybacking*) is to dynamically speed up and slow down client processing rates (e.g., display rates for video files) so as to bring different streams to the same file position, at which time the streams can be merged [GoLM95, AgWY96a, LaLG98]. An appealing aspect of these approaches is that they require the minimum possible client receive bandwidth (i.e., equal to the file play rate[3]) and minimal client buffer space. On the other hand, if clients have receive bandwidth greater than the file

* This work was partially supported by the NSF (Grants CCR-9704503 and CCR-9975044) and NSERC (Grant OGP-0000264). A shorter version of this paper appears in *Proc. 5th Int'l. Workshop on Multimedia Information Systems (MIS '99)*, Indian Wells, CA, Oct. 1999.

[1] More generally, the delivery techniques we consider may be fruitful for any data stream that clients process sequentially.
[2] We use the term "multicast" to denote both multicast and true broadcast throughout this paper.
[3] Throughout the paper we use the term "play rate" to denote the fixed rate at which a file must be transmitted in order for the client to process or play the stream as it arrives. Data is assumed to be transmitted at this rate unless otherwise stated.

To appear in *IEEE Transactions on Knowledge and Data Engineering*, 2001
©2001 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

play rate, and some spare buffer space, significantly greater server bandwidth savings can be achieved [AgWY96b, ViIm96, CaLo97, HuSh97, JuTs98, HuCS98, EaVe98, CaHV99, PaCL99, GaTo99, SGRT99, EaFV99]. In these stream merging methods, a client receiving a particular stream *simultaneously* receives and buffers another portion of the data from a different (multicast) stream, thus enabling greater opportunities for one client to catch up with and share future streams with another client.

Two of these recent techniques, namely dynamic skyscraper (with channel stealing) [EaVe98] and stream tapping/patching/controlled multicast [CaLo97, HuCS98, CaHV99, GaTo99, SGRT99], have the key property that they can provide *immediate* real-time streaming to each client without requiring initial portions of the file to be pre-loaded at the client. These two techniques also require client receive bandwidth at most two times the file play rate. To our knowledge, how these two techniques compare with respect to required server bandwidth has not previously been studied. This paper addresses this issue as well as the following open questions:

(1) What is the minimum required server (disk and network I/O) bandwidth for delivery techniques that provide immediate service to clients?
(2) What is the interplay between achievable server bandwidth reduction and client receive bandwidth?
(3) Are there new (practical) delivery techniques that achieve better bandwidth savings than the previous techniques, yet still provide immediate service to each client request?
(4) How does the best of the techniques that provide immediate service to client requests compare to the static periodic broadcast techniques (e.g., [AgWY96b, ViIm96, HuSh97, JuTs98]) that have fixed server bandwidth independent of the client request rate?

The principal system design results, in order of their appearance in the remainder of this paper, are as follows:

- We review the optimized stream tapping/grace patching/controlled multicast method, for which required server bandwidth increases with the *square root* of the request arrival rate. This is significantly better than when immediate service is provided and multicast delivery is not employed, in which case required server bandwidth increases *linearly* with the request arrival rate.
- We develop a new implementation of the *partitioned* dynamic skyscraper technique [EaFV99] that provides immediate service to client requests more simply and directly than the original dynamic skyscraper method, and we show how to optimize this partitioned dynamic skyscraper architecture.
- The optimized dynamic skyscraper technique has required server bandwidth that increases *logarithmically* (with constant factor between two and three) as a function of the client request rate. Thus, at moderate to high client request rate, the dynamic skyscraper technique significantly outperforms optimized patching/ stream tapping.
- We derive a tight lower bound on the required server bandwidth for any technique that provides immediate service to client requests. This lower bound increases *logarithmically with a constant factor of one* as a function of the client request arrival rate. Thus, techniques that provide immediate service to each client request have the potential to be quite competitive with the static broadcast techniques that have fixed server bandwidth independent of client request rate.
- We define a new family of segmented delivery techniques, called *segmented send-latest receive-earliest* (SSLRE). Although not necessarily practical to implement, the SSLRE techniques demonstrate that it is at least theoretically possible to achieve nearly the lower bound on required server bandwidth if client receive bandwidth equals twice the file play rate, assuming clients can buffer the required data from shared streams.
- We propose a new practical delivery technique, *hierarchical multicast stream merging (HMSM)*, that is simple to implement and provides immediate real-time service to clients. Simulation results show that if client receive bandwidth equals two times the file play rate, the required server bandwidth for the HMSM technique is reasonably close to the minimum achievable required server bandwidth over a wide range of client request rates.

For the purposes of obtaining the lower bound and examining the fundamental capabilities of the various delivery techniques, the above results are obtained assuming that (1) clients have sufficient space for buffering the data streams, and (2) the entire file is consumed sequentially by the client without use of interactive functions such as pause, rewind or fast forward. However, each of the techniques that we consider can be adapted for limited client buffer space, and for interactive functions, with a concomitant increase in required server bandwidth, as discussed in Section 5.

Table 1: Notation

Symbol	Definition
λ_i	request rate for file i
T_i	total time to play file i (equals total time to transmit file i if r = 1)
N_i	average number of requests for file i that arrive during a period of length T_i ($N_i = \lambda_i T_i$)
B_z	required server bandwidth to deliver a particular file using delivery technique z, in units of the file play rate
y_i	threshold for file i in optimized stream tapping/grace patching, expressed as a fraction of T_i
K	number of file segments in dynamic skyscraper
W	largest segment size in dynamic skyscraper
r	stream transmission rate, measured in units of the play rate; default value is r = 1
n	client receive bandwidth, measured in units of the play rate

In this paper, *required* server bandwidth is defined as the *average* server bandwidth used to satisfy client requests for a particular file with a given client request rate, when server bandwidth is unlimited. There are at least two reasons for believing that this single-file metric, which is relatively easy to compute, is a good metric of the server bandwidth needed for a given client load. First, although the server bandwidth consumed for delivery of a given file will vary over time, the *total* bandwidth used to deliver a reasonably large number of files will have lower coefficient of variation over time, for independently requested files and fixed client request rates. Thus, the sum over all files of the average server bandwidth used to deliver each file should be a good estimate of the total server bandwidth needed to achieve very low client waiting time. Second, simulations of various delivery techniques have shown that with fixed client request rates and finite server bandwidth equal to the sum of the average server bandwidth usage for each file, average client waiting time (due to temporary server overload) is close to zero (*e.g.*, [EaFV99, EaVZ99]). Furthermore, the results have also shown that if total server bandwidth is reduced below this value, the probability that a client cannot be served immediately and the average client wait rapidly increase.

The rest of this paper is organized as follows. Section 2 reviews and derives the required server bandwidth for the optimized stream tapping/grace patching/controlled multicast technique. Section 3 reviews the dynamic skyscraper technique, develops the simpler method for providing immediate service to client requests, defines how to optimize the new dynamic skyscraper method, and derives the required server bandwidth. Section 4 derives the lower bound on required server bandwidth for any delivery technique that provides immediate service to clients and shows the impact of client receive bandwidth on this lower bound. Section 5 defines the new hierarchical multicast stream merging technique, and Section 6 concludes the paper.

Table 1 defines notation used throughout the rest of the paper.

2 Required Server Bandwidth for Optimized Stream Tapping/Grace Patching

Two recent papers propose very similar data delivery techniques, called *stream tapping* [CaLo97] and *patching* [HuCS98], which are simple to implement. The best of the proposed patching policies, called *grace patching*, is identical to the stream tapping policy if client buffer space is sufficiently large, as is assumed for comparing delivery techniques in this paper. The optimized version of this delivery technique [CaHV99, GaTo99], which has also been called *controlled multicast* [GaTo99], is considered here.

The stream tapping/grace patching policy operates as follows. In response to a given client request, the server delivers the requested file in a single multicast stream. A client that submits a new request for the same file sufficiently soon after this stream has started begins listening to the multicast, buffering the data received. Each such client is also provided a new unicast stream (i.e., a "patch" stream) that delivers the data that was delivered in the multicast stream prior to the new client's request. Both the multicast stream and the patch stream deliver data at the file play rate so that each client can play the file in real time. Thus, the required client receive bandwidth is twice the file play rate. The patch stream terminates when it reaches the point that the client joined the full-file multicast.

To keep the unicast patch streams short, when the fraction of the file that has been delivered by the most recent multicast exceeds a given threshold, the next client request triggers a new full-file multicast. Let y_i denote the threshold for file i, and T_i denote the duration of the full-file multicast. Assuming Poisson client request arrivals at rate λ_i, the required server bandwidth for delivery of file i using stream tapping/grace patching, measured in units of the play rate, is given by

$$B_{\text{patching}} = \frac{T_i + \lambda_i y_i T_i \dfrac{y_i T_i}{2}}{y_i T_i + \dfrac{1}{\lambda_i}} = \frac{1 + \dfrac{y_i^2 N_i}{2}}{y_i + \dfrac{1}{N_i}},$$

where $N_i = \lambda_i T_i$ is the average number of client requests that arrive during time T_i. The denominator in the middle of the above equation is the average time that elapses between successive full-file multicasts, i.e., the duration of the threshold period plus the average time until the next client request arrives. The numerator is the expected value of the sum of the transmission times of the full-file and patch streams that are initiated during that interval. Note that the average number of patch streams that are started before the threshold expires is $\lambda_i y_i T_i$ and the average duration of the patch streams is $y_i T_i/2$.

Differentiating the above expression with respect to y_i, and setting the result to zero, we obtain $(\sqrt{2N_i+1}-1)/N_i$ as the optimal threshold value. Substituting this value of y_i into the above expression for required server bandwidth yields the following result for the required server bandwidth for optimized stream tapping/grace patching:

$$B_{\text{optimized patching}} = \sqrt{2N_i + 1} - 1. \tag{1}$$

Note that the required server bandwidth grows with the *square root* of the client request rate for the file, and that the optimal threshold decreases as the client request rate increases. (The reader is referred to [GaTo99] for an alternate derivation of these results.)

3 Dynamic Skyscraper Delivery

Section 3.1 reviews the recent partitioned dynamic skyscraper delivery technique [EaFV99] and defines a particular implementation that provides immediate service to client requests in a simpler and more direct way than the original dynamic skyscraper method. The required server bandwidth for this version of partitioned dynamic skyscraper delivery is derived in Section 3.2.

3.1 Providing Immediate Service Using Partitioned Dynamic Skyscraper

The static *skyscraper broadcast* scheme defined in [HuSh97] divides a file into K increasing-sized segments (s_1, s_2, s_3, …, s_K), with largest segment size denoted by W.[4] In a broadband satellite or cable network, each segment is continuously broadcast at the file play rate on its own channel, as illustrated in Figure 1. Each client is given a schedule for tuning into each of the K channels to receive each of the file segments. For example, a client who requests the file just before the broadcast period labelled 3 on the first channel, would be scheduled to receive segments 1-3 sequentially during the periods that are labelled 3, and segments 4-6 during the periods labelled 1 on channels 4-6. The structure of the server transmission schedule ensures that, for any given segment 1 broadcast that a client might receive, the client can receive each other file segment at or before the time it needs to be played, by listening to at most two channels simultaneously. (The reader can verify this

[4] The segment sizes are not strictly increasing, but have the pattern 1,2,2,j,j,k,k,..., with each size change being an increase in size. In the original skyscraper scheme, the progression was specified as 1,2,2,5,5,12,12,25,25,..., upper bounded by the parameter W. This progression appears to have the maximum possible size increases (among progressions with the above pattern) such that clients never need to listen to more than two streams simultaneously. The progression 1,2,2,4,4,8,8,..., provides similar performance for skyscraper broadcasts (i.e., same client receive bandwidth, similar segment 1 delivery time, and similar client buffer space requirement), and increases the efficiency of dynamic skyscraper broadcasts.

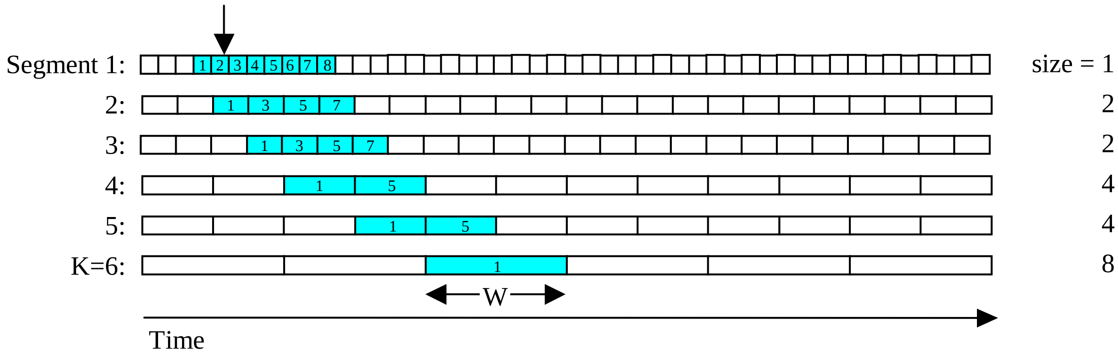

Figure 1: Skyscraper and Dynamic Skyscraper Delivery
(K=6, W=8, segment size progression = 1,2,2,4,4,8)

in Figure 1.) Since larger segments are multicast less frequently, clients must be able to receive and buffer segments ahead of when they need to be played, thus merging with other clients that may be at different play points. The maximum client buffer space needed by any client transmission schedule is equal to the largest segment size (W) [HuSh97].

The required server bandwidth for this static skyscraper method is equal to K, independent of the client request rate for the file. The duration of each segment 1 broadcast is determined by the total file delivery time (T_i) divided by the sum of the segment sizes. Thus, larger values of K and W result in lower average and maximum client wait time for receiving the first segment. A desirable configuration might have K=10 and the segment size progression equal to 1,2,2,4,4,8,8,16,16,32, in which case segment 1 broadcasts begin every 0.01075 T_i , and required server bandwidth for skyscraper is lower than for optimized patching if $N_i > 61$.

The *dynamic skyscraper* delivery technique was proposed in [EaVe98] to improve the performance of the skyscraper technique for lower client request rates and for time-varying file popularities. In this technique, if a client request arrives prior to the broadcast period labeled 1 on the first channel in Figure 1, the set of segment broadcasts that are shaded in the figure (called a *transmission cluster*) might be scheduled to deliver the segments of the file. The arriving client only needs the segment transmissions labeled 1 on each channel. However, any client who requests the same file prior to the transmission period labeled 8 on channel 1 will receive broadcasts from the cluster that was scheduled when the first request arrived. When the broadcast labeled 8 is complete, the six channels (e.g., satellite or cable channels) can be scheduled to deliver an identically structured transmission cluster for a *different file*. Thus, if there is a queue of pending client requests, the six channels will deliver a cluster of segment broadcasts for the file requested by the client at the front of the queue. If no client requests are waiting, the six channels remain idle until a new client request arrives, which then initiates a new transmission cluster. Note that as with optimized patching any queueing discipline, for example one tailored to the needs of the service provider and clients, can be used to determine the order in which waiting clients are served during periods of temporary server overload.

To employ the dynamic skyscraper technique over the Internet, the transmission cluster can be implemented with W multicast streams of varying duration, as shown by the numbering of the cluster transmission periods in Figure 1. That is, the first stream delivers all K segments, the second stream starts one unit segment later and delivers only the first segment, the third stream starts one unit segment later than the second stream and delivers the first three segments, and so on. Total server bandwidth is allocated in units of these clusters of W variable-length streams. Each cluster of streams uses K units of server bandwidth, with each unit of bandwidth used for duration W. Note that this implementation requires clients to join fewer multicast groups and provides more time for joining the successive multicast groups needed to receive the entire file than if there are K multicast groups each transmitting a different segment of the file.

In the original dynamic skyscraper technique [EaVe98], immediate service is provided for clients that are waiting for the start of a segment 1 multicast in a transmission cluster using a technique termed *channel stealing*. That is, (portions of) transmission cluster streams that have no receiving clients may be reallocated to provide quick service to newly arriving requests.

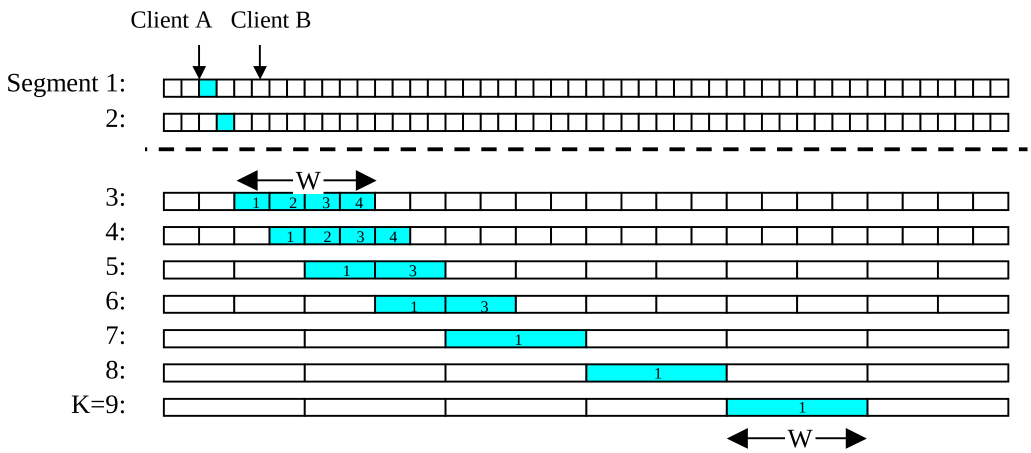

Figure 2: Partitioned Dynamic Skyscraper with Immediate Service
(K=9, W=8, segment size progression = 1,1,2,2,4,4,8,8,8)

We propose a more direct way to provide immediate service to client requests that is also simpler to implement. This more direct approach involves a small modification to the segment size progression, and a particular implementation of *partitioned* skyscraper delivery [EaFV99]. The new segment size progression has the form 1,1,2,2,j,j,k,k,…. As in [EaVe98, EaFV99], each size increase is either two-fold or three-fold, and no two consecutive size increases is three-fold, to limit the number of streams a client must listen to concurrently. Also as before, the progression is upper-bounded by the parameter W, the purpose of which is to limit the required client buffer space.

We partition the segments, as illustrated in Figure 2, so that streams that deliver the first two segments can be scheduled independently and immediately in response to each client request. Thus, when client A requests the file, the server schedules one stream to deliver the first two segments, and schedules W/2 multicast streams to deliver a transmission cluster of segments 3 through K. These streams (one that delivers the first two segments and four that deliver the transmission cluster) are shaded in Figure 2. When client B requests the file, the server only allocates a new stream (from unscheduled bandwidth not shown in the figure) to deliver the first two segments of the file to client B. Client B receives segments 3 through K by listening to the streams in the transmission cluster that was scheduled when client A arrived. The next section computes the average server bandwidth used when the initial two-segment stream and possible new transmission cluster are scheduled immediately for each client request.

A key observation is that this partitioned dynamic skyscraper system with each segment size increase being a two-fold increase (i.e., progression 1,1,2,2,4,4,8,8,…) requires client receive bandwidth equal to only twice the file play rate. The partitioned system with at least one three-fold segment size increase (such as the progression 1,1,2,2,6,6,12,12,36,36,…) requires client receive bandwidth equal to three times the file play rate, as in the corresponding dynamic skyscraper system without partitioning.

3.2 Required Server Bandwidth for Dynamic Skyscraper

Let U denote the duration of a unit-segment multicast, which is determined by the time duration of the entire file (T_i) and the sum of the segment sizes. For the partitioned dynamic skyscraper system defined above, the required server bandwidth for delivery of a file i (measured in units of the streaming rate), given a Poisson request arrival stream, is given by

$$B_{\text{dynamic skyscraper}} = 2U\lambda_i + \frac{(K-2)WU}{WU + \frac{1}{\lambda_i}} = 2U\lambda_i + \frac{K-2}{1+\frac{1}{\lambda_i WU}}. \qquad (2)$$

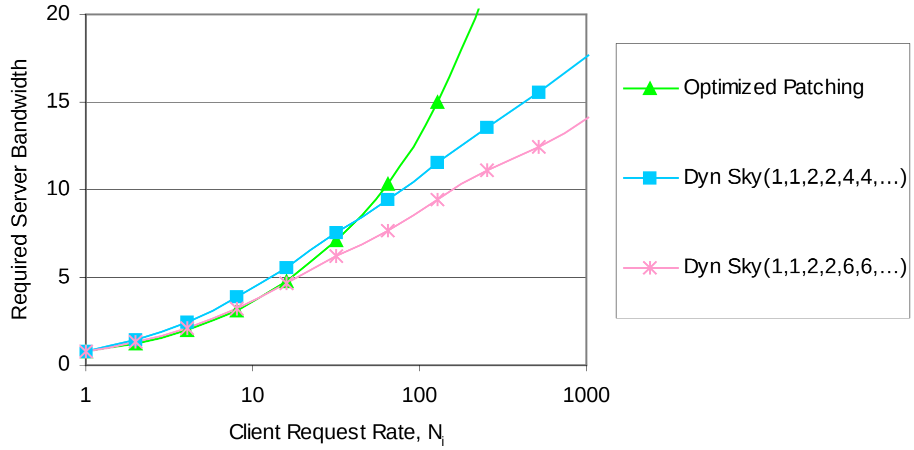

Figure 3: Required Server Bandwidth for Dynamic Skyscraper & Stream Tapping/Patching

The first term is the required bandwidth for delivering the first two unit segments of the file, which involves sending a stream of duration 2U at frequency λ_i. The second term is the required bandwidth for delivering the transmission clusters for the rest of the file, which use K–2 units of server bandwidth, where each unit of bandwidth is used for time equal to WU.

The values of K and W that minimize the required server bandwidth given in equation (2), may be found numerically for any particular segment size progression of interest.

It is also possible to determine the asymptotic behavior for high client request rate. For the progression 1,1,2,2,4,4,8,8,…, Appendix A shows that for $N_i > 128$, the required server bandwidth is approximately

$$B_{1,1,2,2,4,4,8,8,...} \approx 2.885\ln(N_i + 3) - 2.3. \quad (3)$$

Note that the required server bandwidth grows only *logarithmically* with client request rate.

Other skyscraper systems may be similarly analyzed. For the progression 1,1,2,2,6,6,12,12,36,36,…, the required server bandwidth for large client request rate can be shown to be

$$B_{1,1,2,2,6,6,12,12,36,36,...} \approx 2.23\ln(N_i + 2.2) - 0.77. \quad (4)$$

Figure 3 shows the required server bandwidth as a function of client request rate (N_i) for optimized dynamic skyscraper systems with two different segment size progressions, and for optimized stream tapping/grace patching. The required server bandwidth for the skyscraper systems is computed using equation (2) with optimal choices of K and W determined numerically for each N_i. (Recall that the segment size progression 1,1,2,2,4,4,8,8,... has the same client receive bandwidth requirement as optimized stream tapping/patching, namely, two times the file play rate.)

The results show that for N_i greater than 64 requests on average per time T_i, the optimized dynamic skyscraper systems have significantly lower bandwidth requirements than optimized stream tapping/patching. More specifically, the optimized dynamic skyscraper delivery system has required server bandwidth that is reasonably competitive with optimized stream tapping/patching at low client request rate, and is better than or competitive with static broadcast techniques over the range of client request rates shown in the figure.

Skyscraper systems in which a second partition is added between channels k and k+1, 2<k<K–2, are also of interest [EaFV99]. Analysis shows that adding the second partition does not alter the asymptotic behavior under high client request arrival rates, but does improve performance somewhat for more moderate arrival rates, at the cost of an increase in the required client receive bandwidth. (For k odd, client receive bandwidth must be three times the file play rate if the progression is 1,1,2,2,4,4,8,8,…, or four times the file play rate if the progression is 1,1,2,2,6,6,12,12,36,36,….)

4 Minimum Required Server Bandwidth for Immediate Service

Given the results in Figure 3, a key question is whether there exist multicast delivery methods that can significantly outperform optimized stream tapping/patching and dynamic skyscraper, over the wide range of client request arrival rates. In other words, how much further improvement in required server bandwidth is possible? Section 4.1 addresses this question by deriving a very simple, yet tight, lower bound on the required server bandwidth as a function of client request rate, for *any* delivery technique that provides immediate real-time streaming to clients. The lower bound assumes clients have unlimited receive bandwidth. Section 4.2 considers how much the lower bound increases if client receive bandwidth is only equal to n times the file play rate, for arbitrary $n > 1$.

4.1 Lower Bound on Required Server Bandwidth

The lower bound on required server bandwidth for delivery techniques that provide immediate real-time service to clients is derived below initially for Poisson client request arrivals, and then is extended to a much broader class of client request arrival processes.[5]

As before, let $\mathrm{T_i}$ be the duration of file i and λ_i be its average request rate. Consider an infinitesimally small portion of the file that plays at arbitrary time x relative to the beginning of the file. For an arbitrary client request that arrives at time t, this portion of the file can be delivered as late as time $t+x$, but no later than $t+x$, if the system provides immediate real time file delivery to each client. If the portion is multicast at time $t+x$, all other clients who request file i between time t and $t+x$, can receive this multicast of the portion at position x. For Poisson arrivals, the average time from $t+x$ until the next request arrives for file i is $1/\lambda_\mathrm{i}$. Thus, the minimum frequency of multicasts of the portion beginning at position x, under the constraint of immediate real time service to each client, is $1/(x+1/\lambda_\mathrm{i})$. This yields a lower bound on the required server bandwidth, in units of the file play rate, for any technique that provides immediate service to client requests, of

$$\mathrm{B_{minimum}} = \int_0^{\mathrm{T_i}} \frac{\mathrm{d}x}{x+\dfrac{1}{\lambda_\mathrm{i}}} = \ln(\mathrm{T_i}\lambda_\mathrm{i}+1) = \ln(\mathrm{N_i}+1)\,, \tag{5}$$

where $\mathrm{N_i} = \lambda_\mathrm{i}\mathrm{T_i}$ is the average number of requests for the file that arrive during a period of length $\mathrm{T_i}$.

The above lower bound implies that, for Poisson arrivals and immediate real-time service to each client, the required server bandwidth must grow at least logarithmically with the client request rate. In fact, this is true for any client arrival process such that the expected time until the next arrival, conditioned on the fact that some previous request arrived at the current time minus x for any $0 < x < \mathrm{T_i}$, is bounded from above by c/λ_i for some constant c. In this case, we replace $1/\lambda_\mathrm{i}$ in the denominator of the integral in equation (5) with c/λ_i, which yields a lower bound on required server bandwidth of $\ln(\mathrm{N_i}/c+1)$. This result is very similar to that in equation (5); in fact, $\ln(\mathrm{N_i}+1)-\ln(\mathrm{N_i}/c+1) < \ln c$, a constant independent of $\mathrm{N_i}$. Furthermore, this more general lower bound on required server bandwidth is tight if arrivals occur in "batches", with c requests per batch, and the batch arrival times are Poisson with rate λ_i/c. This illustrates the key point that there are greater opportunities for stream sharing, and therefore the server bandwidth requirement is lower, if arrivals are more bursty (i.e., for larger values of c). By considering Poisson arrivals, we can expect conservative performance estimates if the actual client request arrival process is more bursty than the Poisson.

[5] The Poisson assumption is likely to be reasonably accurate for full streaming media file requests [AKEV01], although one would expect the (approximately Poisson) request rate to be time-varying, and in particular, time-of-day dependant. For relatively short files, the Poisson analysis is directly applicable. For the case that substantive change in request rate occurs on a time scale similar to the file play duration (e.g., perhaps for a two hour movie), the analysis for the broader class of arrival processes yields applicable intuition and a similar result.

Further, it is easy to see that the bound in equation (5) can be reformulated as a function of the start-up delay d in a static broadcast scheme, instead of as a function of the client request rate, simply by replacing N_i by T_i / d. This result is shown formally in parallel work by Birk and Mondri [BiMo99].

Comparing equation (5) with equations (1) through (4) shows that there is considerable room for improvement over the optimized stream tapping/patching and dynamic skyscraper delivery methods. However, the lower bound in equation (5) assumes clients can receive arbitrarily many multicasts simultaneously. The next section considers the likely lower bound on required server bandwidth if client receive bandwidth is a (small) multiple of the file play rate.

4.2 Impact of Limited Client Receive Bandwidth

For clients that have receive bandwidth equal to n times the file play rate, for any $n > 1$, we define a new family of delivery techniques. These techniques may not be practical to implement, but intuitively they are likely to require close to the minimum possible server bandwidth for immediate real-time service to clients who have the specified receive bandwidth. Each technique in the family is distinguished by parameters n and r, where r is the stream transmission rate, in units of the file play rate. (In each of the techniques discussed in previous sections of this paper, r is equal to one.) We derive a close upper bound on the required server bandwidth, for any n and r. The results show that for $n = 3$ and $r = 1$, or for $n = 2$ and $r = \varepsilon << 1$, the required server bandwidth for the new technique is nearly equal to the lower bound on required server bandwidth that was derived in Section 4.1 for any technique that provides immediate real-time service to clients. This suggests that it may be possible to develop new practical techniques that nearly achieve the lower bound derived in Section 4.1, and that require client bandwidth of at most two times the play rate.

The new family of delivery techniques operate as follows when the segment transmission rate $r = 1$ (in which case n is an integer). A file is divided into arbitrarily small segments and the following two rules are used to deliver the segments to a client who requests the given file at time t:

1. The client receives any multicast of a segment that begins at a position x in the file, as long as that multicast commences between times t and $t+x$, and as long as receiving that multicast would not violate the limit n on the client receive bandwidth. If at any point in time there are more than n concurrent multicasts that the client could fruitfully receive, the client receives those n segments that occur earliest in the file.
2. Any segment of the file that cannot be received from an existing scheduled multicast is scheduled for multicast by the server at the latest possible time. That is, if the segment begins at position x and is transmitted at the file play rate (i.e., $r = 1$), the segment is scheduled to begin transmission at time $t+x$.

For lack of a better name, we call this family of techniques, and its generalization for $r \neq 1$given in Appendix C, *segmented send-latest receive-earliest* (SSLRE(n,r)). Note that the SSLRE(∞,1) technique achieves the lower bound on required server bandwidth derived in Section 4.1 to any desired precision by dividing the file into sufficiently small segments. This demonstrates that the lower bound derived in the previous section is tight. A similar delivery technique defined only for unlimited client receive bandwidth, but augmented for limited client buffer space, has been shown in parallel work [SGRT99] to be optimal for the case in which available client buffer space may limit which transmissions a client can receive.

The SSLRE(n,r) technique may be impractical to implement, as it results in very fragmented and complex delivery schedules. Furthermore, the SSLRE(n,r) technique does not have minimum required server bandwidth for finite n because there are optimal rearrangements of scheduled multicast transmissions when a new client request arrives that are not performed by SSLRE. However, the required server bandwidth for the SSLRE(n,r) technique, derived below for arbitrary n, provides an upper bound on the minimum required server bandwidth for each possible client receive bandwidth. We speculate that this bound provides accurate insight into the lower bound on required server bandwidth for each client receive bandwidth. In particular, the optimal rearrangements of scheduled multicast transmissions that SSLRE does not perform are, intuitively, likely to have only a secondary effect on required server bandwidth, as compared with the heuristics given in rules 1 and 2 above that are implemented in the SSLRE technique. This intuition is reinforced by the results below that show that the required server bandwidth for SSLRE(3,1), or for SSLRE(2,ε), is very close to the lower bound derived in Section 4.1. (That is, the optimal rearrangements of scheduled multicasts have at most very minor impact on required server bandwidth in these cases.)

For a division of file i into sufficiently many small segments, Appendix B derives the following estimate of the required server bandwidth for the SSRLE(n,1) technique, in units of the file play rate:

$$B_{SSLRE(n,1)} \approx \eta_{n,1} \ln\left(\frac{N_i}{\eta_{n,1}} + 1 \right), \tag{6}$$

where $\eta_{n,1}$ is the positive real constant that satisfies the following equation:

$$\eta_{n,1}\left(1 - \left(\frac{\eta_{n,1}}{\eta_{n,1}+1} \right)^n \right) = 1 .$$

The above result assumes Poisson arrivals, but can be generalized in a similar fashion as was done for the lower bound with unlimited client receive bandwidth. $\eta_{n,1}$ decreases monotonically in n between $\eta_{2,1} = (1+\sqrt{5})/2$ ≈ 1.62 and $\eta_{\infty,1} = 1$. Thus, if the server transmits each segment at the file play rate (i.e., $r = 1$) and if client receive bandwidth is equal to twice the play rate (i.e., $n = 2$), it is possible (although not necessarily practical) to provide immediate service with no more than 62% greater server bandwidth than is minimally required when clients have unbounded receive bandwidth. Further, since $\eta_{3,1} \approx 1.19$, it is possible to achieve nearly all of the benefit of unbounded client bandwidth, with respect to minimizing required server bandwidth, when $r = 1$ and clients have receive bandwidth equal to just three times the play rate.

Appendix C defines the SSLRE technique for the more general case that the segment transmission rate, r, can be different than the file play rate. In this case, a client can listen to at most s concurrent segment transmissions (each at rate r), and total client receive bandwidth, $n = s \times r$, may not be an integer. For this more general family of techniques, Appendix C derives an estimate of the required server bandwidth as

$$B_{SSLRE(n,r)} \approx \eta_{n,r} \ln\left(\frac{N_i}{\eta_{n,r}} + 1 \right), \tag{7}$$

where $\eta_{n,r}$ is the positive real constant that satisfies the following equation:

$$\eta_{n,r}\left(1 - \left(\frac{\eta_{n,r}}{\eta_{n,r}+r} \right)^{\frac{n}{r}} \right) = 1 .$$

For fixed n, the value of $\eta_{n,r}$, and thus the required server bandwidth, is minimized for r tending to zero (i.e., low-rate segment transmissions). Denoting the value of $\eta_{n,r}$ in this limiting case by $\eta_{n,\varepsilon}$, we have

$$\eta_{n,\varepsilon}\left(1 - e^{-n/\eta_{n,\varepsilon}} \right) = 1 .$$

Note[6] that $\eta_{n,\varepsilon}$ decreases monotonically in n, from $\eta_{1,\varepsilon} = \infty$ to $\eta_{\infty,\varepsilon} = 1$, with $\eta_{1.2,\varepsilon} \approx 3.188$ and $\eta_{2,\varepsilon} \approx 1.255$. Thus, for client receive bandwidth n=2, it is (at least theoretically) possible to provide immediate real-time streaming with no more than about 25% greater server bandwidth than is minimally required when clients have unbounded receive bandwidth. Furthermore, for $n < 2$, there is potential for required server bandwidth to grow only logarithmically with a small constant factor as a function of client request rate. Practical techniques that exploit this latter potential are explored in [EaVZ00]. Figure 4 shows the lower bound on the required server bandwidth for unlimited client receive bandwidth from equation (5), and the estimated required server

[6] For $\eta_{1,\varepsilon} = \infty$, equation (7) yields $B_{SSLRE(1,\varepsilon)} = N_i$, which is the expected result.

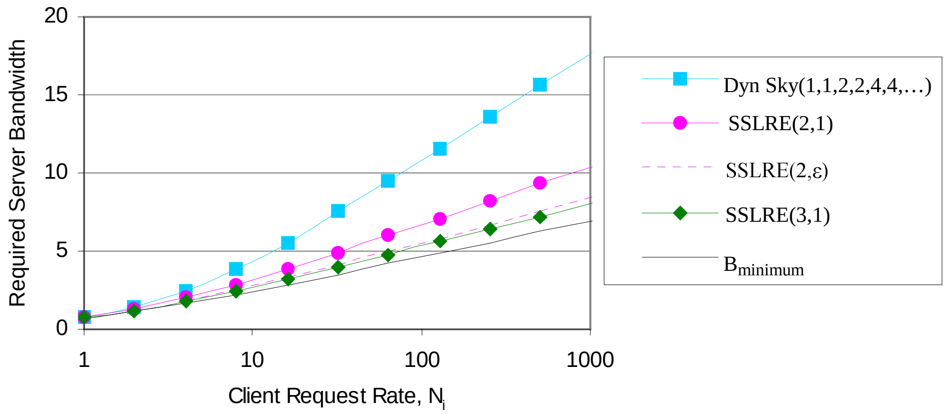

Figure 4: Required Server Bandwidth for Immediate Real Time File Delivery

bandwidth for SSLRE(3,1), SSLRE(2,ε) and SSLRE(2,1) from equation (7), as functions of the client request rate, N_i. Note that $B_{SSLRE(3,1)}$ and $B_{SSLRE(2,\varepsilon)}$ are very close to $B_{minimum}$, and that for client request rates up to at least an average of 1000 requests per file play time, even $B_{SSLRE(2,1)}$ is reasonably competitive with static broadcast techniques (which require fixed server bandwidth on the order of five to ten streams).

The SSLRE(n,r) techniques can be extended in a straightforward way to operate with finite client buffer space. However, since the techniques involve very complex server transmission schedules and client receive schedules, the principal value of these techniques is to determine, approximately, the lowest feasible required server bandwidth for providing immediate service to client requests when clients have receive bandwidth equal to *n*. In the next section we propose a new practical delivery technique and then evaluate the performance of the new technique by comparing its required server bandwidth against the required server bandwidth for SSLRE. We then comment on finite client buffer space in the context of the new practical delivery method.

5 Hierarchical Multicast Stream Merging (HMSM)

The results in Figures 3 and 4 show that there is considerable potential for improving performance over the previous optimized stream tapping/patching/controlled multicast technique and over the optimized dynamic skyscraper technique. Motivated by these results, we propose a new delivery technique that we call *hierarchical multicast stream merging (HMSM).*

The new HMSM technique attempts to capture the advantages of dynamic skyscraper [EaVe98] and piggybacking [GoLM95, AgWY96a, LaLG98], as well as the strengths of stream tapping/patching [CaLo97, HuCS98]. In particular, clients that request the same file are repeatedly merged into larger and larger groups, leading to a hierarchical merging structure (as in dynamic skyscraper or piggybacking). Furthermore, clients are merged using dynamically scheduled patch streams (as in stream tapping/patching), rather than using transmission clusters or altering client play rates.

5.1 HMSM Delivery Technique

Key elements of the hierarchical multicast stream merging technique include: (1) each data transmission stream is multicast so that any client can listen to the stream, (2) clients accumulate data faster than their file play rate (by receiving multiple streams and/or by receiving an accelerated stream), thereby catching up to clients that started receiving the file earlier, (3) clients are merged into larger and larger groups, and (4) once two transmission streams are merged, the clients listen to the same stream(s) to receive the remainder of the file.

The hierarchical multicast stream merging technique is illustrated in Figure 5 for a particular set of request arrivals for an arbitrary file, assuming the server transmits streams at the play rate (i.e., r = 1) and clients have receive bandwidth equal to twice the play rate. In this case, denoted by HMSM(2,1), the most efficient way for a client (or group of clients) to merge with an earlier client or group that requested the same file, is to listen to the latter’s transmission stream, as well as one’s own stream. One unit of time on the x-axis corresponds to the

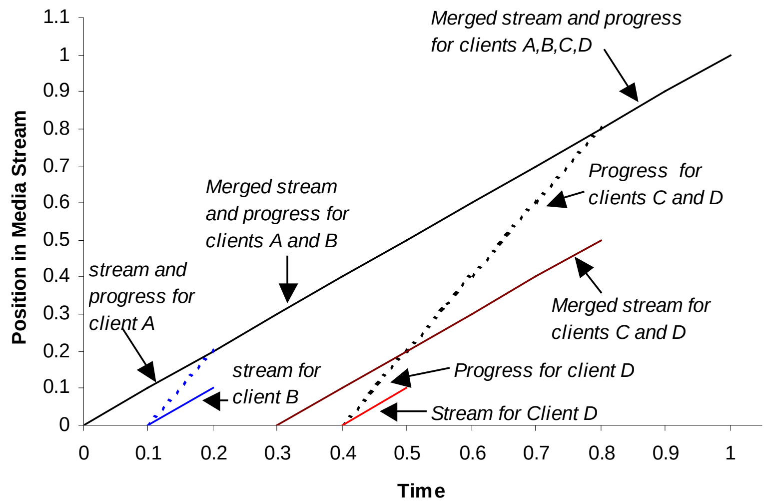

Figure 5: Example of Hierarchical Multicast Stream Merging

total time it takes to deliver the file. One unit of data on the y-axis represents the total data for the file. The solid lines in the figure represent data transmission streams, which always progress through the file at rate equal to one unit of data per unit of time. The dotted lines show the amount of useful data that a client or group of clients has accumulated as a function of time.

In the figure, requests arrive from clients A, B, C, and D at times 0, 0.1, 0.3, and 0.4, respectively. In order to provide immediate service, each new client is provided a new *multicast* stream that initiates delivery of the initial portion of the requested file. Client B also listens to the stream initiated by client A, accumulating data at rate two, and merging with client A at time 0.2. Client D listens to the stream initiated by client C, and merges with client C before client C can merge with client A. When C and D merge, both C and D listen to the streams initiated by A and C until both clients have accumulated enough data to merge with clients A and B.

Note that a hierarchical merging structure would also be formed if clients C and D each listen to and separately merge with the stream initiated by client A. In this case, the merge of clients C and A (which would terminate the stream for C) would take place at time 0.6, and the merge of clients D and A (which would terminate the stream for D) would take place at time 0.8. This alternate hierarchical merging structure, which would occur in the patching technique with threshold larger than 0.4, would require greater server bandwidth for delivering the file.

Variants of hierarchical multicast stream merging differ according to the precise policy used to determine which clients to merge with what others, and in what order, as well as according to what (existing or new) streams are listened to by clients so as to accomplish the desired merges. For homogeneous clients with receive bandwidth equal to twice the streaming rate, [EaVZ99] proposes and evaluates several heuristic policies for merging streams that are transmitted at the file play rate. One particularly simple proposed policy dictates that each client listens to the *closest target* (i.e., the most recently initiated earlier stream that is still active) in addition to its own stream, and that merges occur in time-order (i.e., earliest merge first). The policy evaluations show that this closest target/earliest merge first (CT) policy performs nearly as well as the off-line optimal merging policy (in which client request arrivals are known in advance, streams are delivered at the play rate, and the merges that lead to least total server bandwidth are performed) [EaVZ99]. Hierarchical multicast stream merging policies for homogeneous clients that have receive bandwidth less than twice the play rate are considered in [EaVZ00]. On-going research considers other contexts.

5.2 Required Server Bandwidth for HMSM

Figure 6 provides the required server bandwidth for HMSM(2,1), as obtained from simulation, assuming Poisson arrivals and optimal merges that are computed from known client request arrival times using a dynamic programming technique adapted from [AgWY96a]. As shown in [EaVZ99], there are simple heuristics for

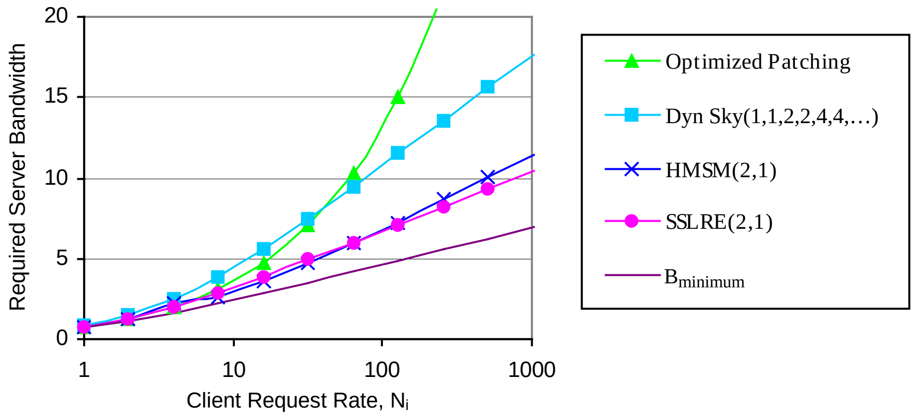

Figure 6: Required Server Bandwidth for Hierarchical Multicast Stream Merging

determining merges with unknown future client request arrival times, such as closest target/earliest merge first, that yield very nearly the same performance as for the offline optimal merges considered here. Also shown in the figure are the lower bound given in equation (5), and the required server bandwidths for SSLRE(2,1), for optimized stream tapping/patching, and for the dynamic skyscraper system with progression 1,1,2,2,4,4,8,8,… and optimal choices of K and W. The results show that HMSM(2,1) yields uniformly good performance, substantially improving on previous techniques. In fact, HMSM(2,1) has nearly identical performance to SSLRE(2,1), which suggests that there is little scope for further improvement for policies that are simple to implement, assuming that $B_{SSLRE(n,1)}$ provides accurate insight into the lower bound on required server bandwidth when streams are transmitted at the file play rate, which seems likely to be the case. Note that the similarity in performance between HMSM(2,1) and SSLRE(2,1) is also perhaps surprising, given the simplicity of the HMSM streams as compared with the complexity of SSLRE segment schedules.

An analytic expression for the required server bandwidth for hierarchical multicast stream merging appears to be quite difficult to obtain. However, for Poisson arrivals and $N_i \leq 1000$, recalling from equation (6) that $B_{SSLRE(2,1)}$ is approximately equal to 1.62 ln (N_i / 1.62 + 1), the results in Figure 6 show that the required server bandwidth for HMSM(2,1) with Poisson arrivals is also reasonably well approximated by 1.62 ln (N_i / 1.62 + 1). Furthermore, an upper bound on the required bandwidth with optimal offline merging, client receive bandwidth equal to twice the file play rate, and an *arbitrary* client request arrival process, derived in Appendix D, is as follows:

$$B_{HMSM-optimaloffline(2,1)} \leq 2.1\ln(\mathrm{N_i}+1)\,.$$

Comparing this upper bound with the approximation for Poisson arrivals shows that the bound is quite conservative for bursty client request arrivals. However, the bound demonstrates that the required server bandwidth for HMSM with client receive bandwidth equal to twice the play rate, is logarithmic in the client request rate for *any* request arrival pattern.

5.3 Finite Buffer Space and Client Interactivity

This paper has thus far discussed and analyzed multicast delivery techniques assuming that clients can buffer all data that is received ahead of its scheduled playback time, and assuming the client does not perform any interactive functions such as pause, rewind, fast forward, skip back, or skip ahead. On the other hand, each of the techniques is easily extended to handle either limited client storage or interactive client requests. For example, Sen et al. have explored how the stream tapping/patching delivery technique should be modified to accommodate constrained client buffer space [SGRT99].

For the HMSM technique, if a client does not have the buffer space to implement a given merge, that particular merge is simply not scheduled. The impact of limited client buffer space on the performance of HMSM is studied in [EaVZ99] for client receive bandwidth equal to twice the play rate, and in [EaVZ00] for client receive bandwidth less than twice the file play rate. Both studies show that, if clients can store 5-10% of the full file and client request arrivals are Poisson, the impact of limited client buffer space on required server bandwidth is fairly small [EaVZ99, EaVZ00]. The intuitive explanation for this result is that when client requests are bursty, buffer space equal to 5-10% of the file enables most of the merges to take place.

The HMSM technique is also easily extended for interactive client requests. For example, with fast forward, the client is given a new multicast stream during the fast forward operation, and when the fast forward operation is complete, the new stream is merged with other streams as in the standard HMSM policy. Similarly, for pause, rewind, skip back/ahead, or other interactive requests, the client is given a new stream at the start or end of the interactive request (as appropriate), and the new stream is merged with other streams at the end of the interactive operation. Disk storage techniques that support many of the interactive functions are discussed, for example, in [SaMR00]. The required server bandwidth for supporting interactive functions depends on the frequency, type, and duration of the interactive requests. The extra server bandwidth needed *during* the interactive requests will be the same for all multicast delivery techniques. Exploring the full impact of client interactivity on the relative required server bandwidth for various delivery techniques is left for future work.

6 Conclusions

This paper has investigated the required server bandwidth for on-demand real-time delivery of large popular data files, assuming that a multicast capability is available so that multiple clients can share reception of a single data transmission. As explained in Section 1, we defined required server bandwidth to be the average server bandwidth used to deliver a file with a given client request rate, when the server has unlimited (disk and network) bandwidth.

We developed a new implementation of the partitioned dynamic skyscraper delivery technique that provides immediate service to clients more simply and easily than the original dynamic skyscraper technique. We defined a method for determining the optimal parameters (i.e., K and W) of this new dynamic skyscraper system for a given client request rate, and derived the required server bandwidth, which is logarithmic with a constant factor between two and three in the client request rate. Thus, at moderate to high client request rates, the dynamic skyscraper system outperforms the optimized stream tapping/patching/controlled multicast technique, which has required server bandwidth that increases with the square root of the client request rate.

We derived a tight lower bound on the server bandwidth required for any technique that provides immediate real-time service, and found that this bandwidth must grow at least logarithmically with the client request rate. By defining and analyzing the required server bandwidth for a new family of delivery techniques with complex and fragmented segment delivery schedules (SSLRE(n,r)), we showed that required server bandwidth generally *increases* as client receive bandwidth (n) *decreases*, but that for client receive bandwidth equal to twice the file play rate there is potential for required server bandwidth to be no more than 25%-62% greater than the lower bound. The results for SSLRE(n,r) also demonstrated that for client receive bandwidth less than twice the file play rate (i.e., $n < 2$) there is potential for required server bandwidth to increase only logarithmically with a small constant factor as client request rate increases.

We proposed a new practical delivery method, hierarchical multicast stream merging (HMSM), which merges clients into larger and larger groups that share multicast streams, without altering the client play rate. This new technique has a number of important advantages. First, it is simple to implement and it is easily extended to operate in the presence of interactive client requests. Second HMSM with $n = 2$ outperforms the optimized stream tapping/patching and optimized dynamic skyscraper techniques at all client request rates, and not much further improvement in required server bandwidth is possible. The HMSM technique with $n = 2$ is also competitive with static broadcast techniques at high client request rates (and is far superior to static broadcast techniques when client request rate is low or varying). For example, if on average 1000 client requests arrive during the time it takes to transmit the full file at the file play rate, the required server bandwidth for providing immediate real-time service to each client using HMSM (n=2) is approximately equal to 10 streams at the file play rate. (Efficient HMSM techniques with n < 2 are defined in [EaVZ00].) Finally, the slow logarithmic increase in required server bandwidth as a function of client request rate implies that the HMSM delivery technique could be used to offer a new service to clients that join a live multicast after it has begun, namely each such client could start at an earlier point in the multicast and then catch up to the live multicast stream, without much increase in the server bandwidth expended on the live multicast. Note also that, like the other multicast delivery methods examined in this paper, the HMSM delivery technique is not dependent on any particular form of multicast support in the network. IP multicast, application-level multicast, broadband satellite or cable broadcast, or other multicast mechanisms can be used to deliver the HMSM data streams.

On-going research includes: (1) designing HMSM policies for composite objects and for clients with heterogeneous receive bandwidths and storage capacities, (2) evaluating the impact of file indexing and client interactive functions on required server bandwidth for HMSM, (3) developing optimal real-time delivery techniques that support recovery from packet loss, (4) developing optimized caching models and strategies [EaFV99, EaFV00] for HMSM systems, (5) designing and evaluating disk load balancing strategies for HMSM systems, and (6) the design and implementation of a prototype system that supports experimental evaluation of alternative delivery techniques and caching strategies.

References

[AgWY96a] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On Optimal Piggyback Merging Policies for Video-On-Demand Systems", *Proc. 1996 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems,* Philadelphia, PA, May 1996, pp. 200-209.

[AgWY96b] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "A Permutation Based Pyramid Broadcasting Scheme for Video-On-Demand Systems", *Proc. IEEE Int'l. Conf. on Multimedia Computing and Systems (ICMCS'96),* Hiroshima, Japan, June 1996.

[AKEV01] J. M. Almeida, J. Krueger, D. L. Eager, and M. K. Vernon, "Analysis of Educational Media Server Workloads", *Proc. 11th Int'l. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2001)*, June 2001.

[BiMo99] Y. Birk and R. Mondri, "Tailored Transmissions for Efficient Near-Video-On-Demand Service", *Proc. IEEE Int'l. Conf. on Multimedia Computing and Systems (ICMCS'99),* Florence, Italy, June 1999, pp. 226-231.

[CaHV99] Y. Cai, K. A. Hua, and K. Vu, "Optimizing Patching Performance", *Proc. IS&T/SPIE Conf. on Multimedia Computing and Networking 1999 (MMCN'99),* San Jose, CA, Jan. 1999, pp. 204-215.

[CaLo97] S. W. Carter and D. D. E. Long, "Improving Video-on-Demand Server Efficiency Through Stream Tapping", *Proc. 6th Int'l. Conf. on Computer Communications and Networks (ICCCN'97)*, Las Vegas, NV, Sept. 1997, pp. 200-207.

[DaSS94] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-demand Video Server with Batching", *Proc. 2nd ACM Int'l. Multimedia Conf. (ACM Multimedia '94),* San Francisco, CA, Oct. 1994, pp. 15-23.

[EaVe98] D. L. Eager and M. K. Vernon, "Dynamic Skyscraper Broadcasts for Video-on-Demand", *Proc. 4th Int'l. Workshop on Multimedia Information Systems (MIS '98)*, Istanbul, Turkey, Sept. 1998, pp. 18-32.

[EaFV99] D. L. Eager, M. C. Ferris, and M. K. Vernon, "Optimized Regional Caching for On-Demand Data Delivery", *Proc. IS&T/SPIE Conf. on Multimedia Computing and Networking 1999 (MMCN'99),* San Jose, CA, Jan. 1999, pp. 301-316.

[EaFV00] D. L. Eager, M. C. Ferris, and M. K. Vernon, "Optimized Caching in Systems with Heterogeneous Client Populations", *Performance Evaluation*, Special Issue on Internet Performance Modeling, Vol. 42, No. 2/3, Sept. 2000, pp. 163-185.

[EaVZ99] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Optimal and Efficient Merging Schedules for Video-on-Demand Servers", *Proc. 7th ACM Int'l. Multimedia Conf. (ACM MULTIMEDIA '99),* Orlando, FL, Nov. 1999, pp. 199-202.

[EaVZ00] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Bandwidth Skimming: A Technique for Cost-Effective Video-on-Demand", *Proc. IS&T/SPIE Conf. On Multimedia Computing and Networking 2000 (MMCN 2000)*, San Jose, CA, Jan. 2000, pp. 206-215.

[GaTo99] L. Gao and D. Towsley, "Supplying Instantaneous Video-on-Demand Services Using Controlled Multicast", *Proc. 1999 IEEE Int'l. Conf. On Multimedia Computing and Systems (ICMCS'99),* Florence, Italy, June 1999.

[GoLM95] L. Golubchik, J. C. S. Lui, and R. Muntz, "Reducing I/O Demand in Video-On-Demand Storage Servers", *Proc. 1995 ACM SIGMETRICS Joint Int'l. Conf. on Measurement and Modeling of Computer Systems,* Ottawa, Canada, May 1995, pp. 25-36.

[HuSh97] K. A. Hua and S. Sheu, "Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems", *Proc. ACM SIGCOMM'97 Conf.,* Cannes, France, Sept. 1997, pp. 89-100.

[HuCS98] K. A. Hua, Y. Cai, and S. Sheu, “Patching: A Multicast Technique for True Video-On-Demand Services”, *Proc. 6th ACM Int'l. Multimedia Conf. (ACM MULTIMEDIA ’98),* Bristol, U.K., Sept. 1998, pp. 191-200.

[JuTs98] L. Juhn and L. Tseng, "Fast Data Broadcasting and Receiving Scheme for Popular Video Service", *IEEE Trans. On Broadcasting 44*, 1 (March 1998), pp. 100-105.

[Klei75] L. Kleinrock. *Queueing Systems: Vol. 1, Theory*. Wiley, New York, 1976.

[LaLG98] S. W. Lau, J. C.-S. Lui, and L. Golubchik, “Merging Video Streams in a Multimedia Storage Server: Complexity and Heuristics”, *ACM Multimedia Systems Journal 6,* 1 (Jan. 1998), pp. 29-42.

[PaCL99] J.-F. Paris, S. W. Carter, and D. D. E. Long, “A Hybrid Broadcasting Protocol for Video On Demand”, *Proc. IS&T/SPIE Conf. on Multimedia Computing and Networking 1999 (MMCN’99),* San Jose, CA, Jan. 1999, pp. 317-326.

[SaMR00] J. R. Santos, R. R. Muntz, and B. Ribeiro-Neto, "Comparing Random Data Allocation and Data Striping in Multimedia Servers", *Proc. ACM SIGMETRICS 2000 Int'l. Conf. on Measurement and Modeling of Computer Systems,* Santa Clara, CA, June 2000, pp. 44-55.

[SGRT99] S. Sen, L. Gao, J. Rexford, and D. Towsley, “Optimal Patching Schemes for Efficient Multimedia Streaming”, *Proc. 9th Int’l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV’99),* Basking Ridge, NJ, June 1999.

[ViIm96] S. Viswanathan and T. Imielinski, “Metropolitan Area Video-on-Demand Service using Pyramid Broadcasting”, *Multimedia Systems 4,* 4 (Aug. 1996), pp. 197-208.

Appendix A: Asymptotic Analysis of $B_{\text{dynamic skyscraper}}$

Here we derive the server bandwidth requirements for the partitioned dynamic skyscraper systems defined in Section 3.1, under high client request rate. The asymptotic analysis that we perform considers the case of λ_i tending to infinity, and then we comment on how large λ_i must be in order for the approximation obtained to be accurate. We assume the segment size progression 1,1,2,2,4,4,8,8,…. Similar analyses can be performed for other progressions.

Recalling equation (2) in Section 3.2, the required server bandwidth (measured in units of the play rate) for delivery of a file i, assuming Poisson client request arrivals, is given by:

$$B_{\text{dynamic skyscraper}} = 2U\lambda_i + \frac{(K-2)}{1+\frac{1}{\lambda_i WU}} \quad . \qquad \text{(A-1)}$$

The first goal is to find the values of K and W (as functions of λ_i) that minimize the required server bandwidth as given by the above expression as λ_i tends to infinity. Note that U is equal to T_i divided by the sum of the segment sizes, and thus $WU \geq T_i / K$. As we show below, there are choices of K and W (as functions of λ_i) such that server bandwidth is asymptotically logarithmic in λ_i. Thus, since the required server bandwidth is greater than or equal to $(K-2)/(1+K/(\lambda_i T_i))$, the optimal value of K must grow more slowly than linearly in λ_i. This implies that, for optimal K, the above expression for required server bandwidth must tend to $2U\lambda_i + (K-2)$ as λ_i tends to infinity. This in turn implies that required server bandwidth is minimized (asymptotically for optimal K) when the value of W is chosen to be as large as possible, and therefore U is as small as possible

For the assumed segment size progression (1,1,2,2,4,4,8,8,...), the maximum value of W for K even is $2^{(K/2)-1}$, while for K odd the maximum value of W is $2^{(K-1)/2}$. For these maximum values of W, the sum of the segment sizes is equal to $2\,(2^{K/2}-1)$ for K even and $3\,2^{(K-1)/2}-2$ for K odd.

Consider, for example, the case of K odd. (We obtain a similar asymptotic result for required server bandwidth for K even.) In this case, $U = T_i/[3\,2^{(K-1)/2}-2] = T_i/(3W-2)$. Note that the optimal value of K (and W) must grow without bound as the client request arrival rate grows without bound, as otherwise, $2U\lambda_i$, and thus the required server bandwidth, would be asymptotically linear in λ_i. Thus, asymptotically U approaches $T_i / 3W$ when W and K are chosen optimally (and assuming that K is odd). Substituting $U = T_i / 3W$ and $W = 2^{(K-1)/2}$ into equation (A-1) for required server bandwidth, and setting the derivative of the resulting expression with respect to K equal to zero, yields

$$\frac{1}{1+3/N_i} - \frac{N_i \ln 2}{3 \times 2^{(K_{optimal}-1)/2}} = 0\,.$$

Solving for $K_{optimal}$ yields the following result:

$$K_{optimal} \approx \frac{2}{\ln 2} \ln\left((N_i + 3) \frac{\ln 2}{3} \right) + 1\,.$$

Substituting this optimal value of K into equation (A-1), with $U = T_i / 3W$ and $W = 2^{(K-1)/2}$, yields

$$B_{1,1,2,2,4,4,8,8,...} \approx \frac{2}{\ln 2} + \frac{2}{\ln 2} \ln(N_i + 3) + \frac{2}{\ln 2} \ln\left(\frac{\ln 2}{3} \right) - 1\,.$$

This reduces to equation (3) in Section 3.2.

Note from the above derivation that this approximation for required server bandwidth will be reasonably accurate if $N_i = \lambda_i T_i$ is large enough such that the optimal value of W is equal to the largest possible value of W for the optimal value of K and if $3W-2 \approx 3W$. The reader can verify that for $N_i > 128$, the approximation in equation (3) yields results that are within 2% of the required server bandwidth computed from equation (2) with optimal K and W determined numerically.

Appendix B: Derivation of $B_{SSLRE(n,1)}$

In the following we determine an estimate (more precisely, a close upper bound) on the required server bandwidth for the SSLRE delivery technique described in Section 4.2. We assume a Poisson arrival process of client requests, although the analysis can be generalized as was done in Section 4.1. Further, we assume that the client receive bandwidth is limited to n (in units of the play rate), and that the file i under consideration is divided into an unbounded number of infinitesimally small segments, each delivered at the play rate ($r = 1$).

Let $u_{i,n,1}(x)$ denote the average duration of the "catch-up window" (period of time during which arriving clients will share in a single multicast transmission) for the infinitesimally small segment of file i at position x. The average required server bandwidth can then be written as

$$B_{SSLRE(n,1)} = \int_0^{T_i} \frac{dx}{u_{i,n,1}(x) + \frac{1}{\lambda_i}}\,.$$

We show below that $u_{i,n,1}(x) \geq \frac{x}{\eta_{n,1}}$, approaching this bound asymptotically for large λ_i, where $\eta_{n,1}$ is the positive real constant satisfying the following equation:

$$\eta_{n,1}\left(1 - \left(\frac{\eta_{n,1}}{\eta_{n,1}+1} \right)^n \right) = 1\,. \qquad \text{(B-1)}$$

Substituting $\frac{x}{\eta_{n,1}}$ for $u_{i,n,1}(x)$ yields the estimate of the required server bandwidth given in equation (6) of Section 4.2.

The lower bound, $\underline{u}_{n,1}(x) = \frac{x}{\eta_{n,1}}$, on the duration of the catch-up window can be derived by considering the case of a continuous stream of arrivals. In this case, there are always at least n segments from earlier in the file that are being multicast concurrently with any multicast M of the segment at position x. Counting back from the segment at position x, the position of the earlier segment that is the n^{th} closest to position x from among those that are being concurrently multicast, determines the duration of the catch-up window.

To illustrate this point, Figure B.1 depicts the operation of the delivery technique for a case with high client request rate and n=2. The figure assumes that the file is divided into 12 equal-sized segments. Each row corresponds to the scheduled multicasts for a distinct segment, with segment position in the file increasing from top to bottom. Each column corresponds to a distinct "time slot", of length equal to the duration of a segment, and with time increasing from left to right. A diagonal line at a particular position denotes a transmission of the corresponding segment during the corresponding time slot. For simplicity of presentation, the figure assumes that the delivery times of the first segment are constrained to begin on time slot boundaries.

Consider the particular transmission of the 8th segment that is indicated by the first darkened diagonal line in the 8th row. The segment that occurs earlier in the file, and that is the 2nd closest within the file to the 8th segment, among those being concurrently multicast, is the 2nd segment. A darkened diagonal line is used in the figure to indicate its concurrent transmission.

Any client that requested the file earlier than the time of the request that triggered the scheduling of the concurrent multicast M_n of the n^{th} earlier segment, and that has not yet received the segment at position x, will receive M. This is since it is unable to receive M_n or any concurrent multicasts of earlier segments, and since it will preferentially receive M over any concurrent multicasts of segments later than that at position x. It is unable to receive M_n owing to the fact that whenever a new segment multicast is scheduled, it is scheduled at the latest possible time given the arrival time of the triggering client request. It preferentially receives M since multicasts of segments that occur earlier in the file are received in preference to concurrent multicasts of segments that occur later in the file, whenever a choice among such segments must be made owing to limited client receive bandwidth.

Finally, the client whose request triggered the scheduling of M_n, and clients making later requests, will not receive M as there are at least n concurrent multicasts of earlier segments that such clients may receive. These segments will not have been previously received by such clients, as the rules defining the delivery technique imply that a subsequent multicast of a segment is scheduled too late to receive for clients that were able to receive the previous multicast of the segment.

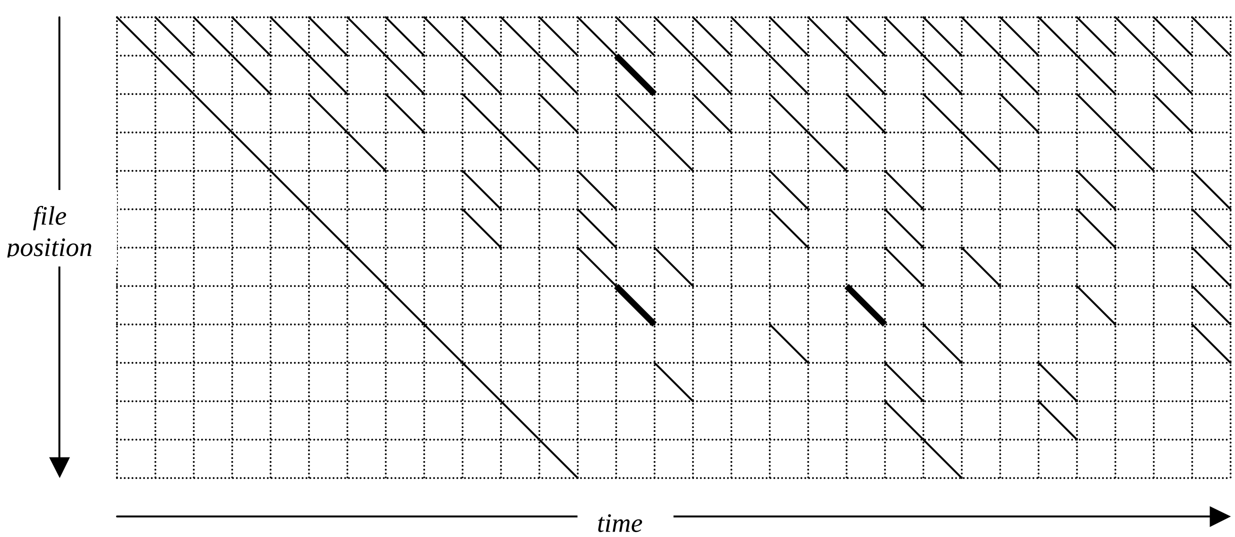

Figure B.1: SSLRE(n,1) Delivery Technique Under High Client Request Rate
(Client Receive Bandwidth is twice the Play Rate)

Suppose that the n^{th} earlier segment is at position x_n in the file. Since both M and M_n were scheduled at the latest possible times given the arrival times of the client requests that initiated these multicasts, the difference in these arrival times (and thus the duration of the catch-up window) must be equal to $x - x_n$. In the example in Figure B.1, $x - x_n = 8–2 = 6$ (in units of the segment transmission time), and the next transmission of the 8^{th} segment occurs 6 time slots later, as indicated by another darkened diagonal line. In general, for the case of an unbounded number of infinitesimally small segments and a continuous stream of arrivals, the average duration of the catch-up window is given by the average value of $x - x_n$ in this context, yielding the following expression for $\underline{u}_{n,1}(x)$:

$$\underline{u}_{n,1}(x) = \int_0^x \int_0^{x_1} \cdots \int_0^{x_{n-1}} (x - x_n) e^{-\int_{x_n}^{x} \frac{dz}{\underline{u}_{n,1}(z)}} \frac{dx_n}{\underline{u}_{n,1}(x_n)} \frac{dx_{n-1}}{\underline{u}_{n,1}(x_{n-1})} \cdots \frac{dx_1}{\underline{u}_{n,1}(x_1)} .$$

Here x_1 denotes the position of the earlier segment among those being concurrently multicast that is closest to position x, x_2 denotes the position of the 2^{nd} closest earlier segment among those being concurrently multicast, and so on. Each term $\frac{dz}{\underline{u}_{n,1}(z)}$ gives the probability that an infinitesimally small segment of duration dz at position z is being multicast concurrently with a segment at position x. The term $e^{-\int_{x_n}^{x} \frac{dz}{\underline{u}_{n,1}(z)}}$ gives the probability that no segments at positions between x_n and x other than possibly those at positions $x_1, x_2, \ldots, x_n$ are being concurrently multicast, and follows from the fact that the sum of a large number of independent stationary renewal processes (each with bounded variance of renewal time) tends to a Poisson process [Klei75], and the fact that the probability of a multicast of any of these specific segments at some specific point in time is infinitesimally small.

As can be verified by substitution and evaluation of the integrals, the solution to the above integral equation is $\underline{u}_{n,1}(x) = \frac{x}{\eta_n}$, where η_n is the positive real constant that satisfies equation (B-1) above.

Appendix C: Definition of SSLRE(n,r) and Derivation of $B_{SSLRE(n,r)}$

In Appendix B, we assume that the streaming rate is equal to the play rate. Here we modify that analysis for the case in which the streaming rate of each segment is less than the play rate ($r < 1$), and the client receive bandwidth n is such that at most some finite integer number n/r of segments can be concurrently received. In this context the delivery technique SSLRE operates as described previously in Section 4.2, excepting with a limit of n/r rather than n on the number of segments that can be concurrently received, and with allowance for the fact that each segment now takes $1/r$ times as long to deliver.[7]

Similarly to the analysis in Appendix B, let $u_{i,n,r}(x)$ denote the average duration of the "catch-up window" (period of time during which arriving clients will share in a single multicast transmission) for the infinitesimally small segment of file i at position x. The required server bandwidth can then be written as

[7] To avoid jitter clients must begin to receive each segment of play duration dx, at least time (dx)/r prior to when it needs to be played. Thus, if immediate service is to be achieved, an initial portion of each file must be delivered at least at the file play rate, instead of at rate r. For example, an initial portion of play duration (dx)n/(r(n-1)) could be delivered at the full client receive bandwidth n, ensuring that if reception of the first segment of the remainder of the file begins immediately after the client has received this initial portion, it can be received in time. Since segments are assumed to be of infinitesimal duration, so is the single initial portion that must be delivered at higher rate, and thus by itself it makes a negligible contribution to bandwidth usage and can be neglected in the analysis.

$$B_{SSLRE(n,r)} = \int_0^{T_i} \frac{dx}{u_{i,n,r}(x) + \frac{1}{\lambda_i}}.$$

We show below that $u_{i,n,r}(x) \geq \frac{x}{\eta_{n,r}}$, approaching this bound asymptotically for large λ_i, where $\eta_{n,r}$ is the positive real constant that satisfies the following equation:

$$\eta_{n,r}\left(1 - \left(\frac{\eta_{n,r}}{\eta_{n,r} + r}\right)^{\frac{n}{r}}\right) = 1. \tag{C-1}$$

Substituting $\frac{x}{\eta_{n,r}}$ for $u_{i,n,r}(x)$ yields the estimate of the required server bandwidth given in equation (7) of Section 4.2.

Similarly as for when segments are delivered at the play rate, as analyzed in Appendix B, the lower bound $\underline{u}_{n,r}(x) = \frac{x}{\eta_{n,r}}$ on the duration of the catch-up window can be derived by considering the case of a continuous stream of arrivals. In this case, there are always at least *n/r* segments from earlier in the file that are being multicast concurrently with any multicast *M* of the segment at position *x*. Counting back from the segment at position *x*, the position of the earlier segment that is the *(n/r)*th closest to position *x* from among those that are being concurrently multicast, determines the duration of the catch-up window.

Suppose that the *(n/r)*th earlier segment is at position $x_{n/r}$ in the file. Since each segment multicast is scheduled at the latest possible time given the arrival time of the client request that initiates it, the duration of the catch-up window is equal to $x - x_{n/r}$. Thus, the average duration of the catch-up window is given by the average value of $x - x_{n/r}$ in this context, yielding the following expression for $\underline{u}_{n,r}(x)$:

$$\underline{u}_{n,r}(x) = \int_0^x \int_0^{x_1} \ldots \int_0^{x_{n/r-1}} (x - x_{n/r}) e^{-\int_{x_{n/r}}^{x} \frac{dz}{r\underline{u}_{n,r}(z)}} \frac{dx_{n/r}}{r\underline{u}_{n,r}(x_{n/r})} \frac{dx_{n/r-1}}{r\underline{u}_{n,r}(x_{n/r-1})} \cdots \frac{dx_1}{r\underline{u}_{n,r}(x_1)}.$$

Here x_1 denotes the position of the earlier segment among those being concurrently multicast that is closest to position x, x_2 denotes the position of the 2nd closest earlier segment among those being concurrently multicast, and so on. Each term $\frac{dz}{r\underline{u}_{n,r}(z)}$ gives the probability that an infinitesimally small segment of size d*z* (and thus transmission duration (d*z*)/*r*) at position *z* is being multicast concurrently with a segment at position *x*. The term $e^{-\int_{x_{n/r}}^{x} \frac{dz}{r\underline{u}_{n,r}(z)}}$ gives the probability that no segments at positions between $x_{n/r}$ and *x* other than possibly those at positions $x_1, x_2, \ldots, x_{n/r}$ are being concurrently multicast.

As can be verified by substitution and evaluation of the integrals, the solution to the above integral equation is $\underline{u}_{n,r}(x) = \frac{x}{\eta_{n,r}}$, where $\eta_{n,r}$ is the positive real constant that satisfies equation (C-1) given above.

Appendix D: Upper Bound on $B_{\text{HMSM-offline optimal}}$ (n=2, r=1)

In the following we derive an upper bound on the required server bandwidth for HMSM with optimal offline merging, assuming the server transmits streams at the file play rate, client receive bandwidth is equal to twice the play rate, and an *arbitrary* client request arrival process.

Clearly, the bandwidth consumed with any specific hierarchical merging policy gives an upper bound on that required with optimal offline hierarchical stream merging. Consider an arbitrary request by client A for a file i, at some time *t*, that results in a full-file multicast (i.e., such that client A is not merged with any earlier requesting client). In the policy that we consider, all clients that make requests within the time interval $[t, t+\mathrm{T_i}/2)$ are eventually merged with client A. The next request for file i that arrives after time $t + \mathrm{T_i}/2$ triggers a new full-file multicast. A group of clients consisting of a client whose request triggers a full-file multicast, together with those clients that are eventually merged with this client, is termed a "cohort" in the following.

Let *c* denote the size of a cohort. We will derive an upper bound, *D*(*c*), on the amount of data that must be multicast in order to serve a cohort of size *c*. (*D(c)* will be measured in units of the data delivered for a full-file multicast.) As will be seen below, *D(c)* is a concave function of the cohort size *c*. This implies that, for a given request arrival rate and average spacing between cohorts, the required server bandwidth that we compute from *D(c)* is maximized when the size of each cohort is identical.[8] Further, for a given arrival rate the required server bandwidth that we compute from *D(c)* is maximized when the spacing between cohorts shrinks to zero; i.e., a new client requests the file at time $\mathrm{T_i}/2$ after the request of the first client in the previous cohort, and becomes the first client in the next cohort. Thus, once we have derived *D(c)*, an upper bound on the required server bandwidth (in units of the play rate) for request rate λ_i, is given by $[D(c)/(\mathrm{T_i}/2)]/(1/\mathrm{T_i}) = 2D(c)$, where $c = \lambda_\mathrm{i}\mathrm{T_i}/2$.

Consider a cohort of integer size *c*. Clearly, we can choose $D(1) = 1$. For $c \ge 2$, the amount of data that must be multicast under hierarchical multicast stream merging is maximized if the request of the latest client occurs at time $(\mathrm{T_i}/2)^{-}$ after the request of the first client. Thus, in the following, we consider only this case.

For $c \ge 3$, our merging policy operates as follows. Assuming that the request from the first client in the cohort was made at time *t*, all those clients whose requests are made in the interval $[t, t+\mathrm{T_i}/4)$ are merged together prior to merging with the rest of the cohort. Correspondingly, all those clients whose requests are made in the interval $[t+\mathrm{T_i}/4, t+\mathrm{T_i}/2)$ are merged together, prior to merging with the other clients. The amount of data that must be multicast with this merging policy is maximized if there is a client whose request occurs at time $t+\mathrm{T_i}/4$. (If there is at least one client, other than the latest client, that makes a request in the interval $[t+\mathrm{T_i}/4, t+\mathrm{T_i}/2)$, it is clear that the quantity of multicast data is maximized if the request of the earliest such client, is made as early as possible within this interval. If there is no such client, the cost would be no less if the request of the latest client in the interval $[t, t+\mathrm{T_i}/4)$ occurred instead at time $t+\mathrm{T_i}/4$, with an associated transmission cost of 1/2 of a full-file multicast.) Thus, in the following, we consider only this case.

For $c \ge 4$, and with the assumption that there is a client that made a request at time $t+\mathrm{T_i}/4$, we refine our merging policy as follows. All those clients whose requests are made in the interval $[t+\mathrm{T_i}/4, t+3\mathrm{T_i}/8)$ are merged prior to merging with the rest of the cohort. Correspondingly, all those clients whose requests are made in the interval $[t+3\mathrm{T_i}/8, t+\mathrm{T_i}/2)$ are merged together, prior to merging with the other clients. The amount of data that must be multicast with this merging policy is maximized if there is a client whose request occurs at time $t+(\mathrm{T_i}/4)^{-}$. (If there is at least one client that makes a request in the interval $[t, t+\mathrm{T_i}/4)$, it is clear that the quantity of multicast data is maximized if the request of the latest such client, is made as late as possible within this interval. If there is no such client, the cost would be no less if the request of the earliest

[8] This may require evaluation of the upper bound for non-integral cohort sizes.

client in the interval $(t+\mathrm{T_i}/4, t+\mathrm{T_i}/2)$ occurred instead at time $t+(\mathrm{T_i}/4)^{-}$ with an associated transmission cost of 1/4 of a full-file multicast.) Thus, in the following, we consider only this case.

For $c \geq 5$, we can recursively apply the same arguments as above, on the "sub-cohort" consisting of the arrivals in the interval $[t, t+\mathrm{T_i}/4)$, and on the sub-cohort consisting of the arrivals in the (identically-sized) interval $[t+\mathrm{T_i}/4, t+\mathrm{T_i}/2)$. Since the amount of data that must be multicast in order to serve a cohort (or sub-cohort) following the above construction is a concave function of the size of the cohort, we need only consider the case in which the size of each sub-cohort differs by at most one.

Consider now a cohort of size $c = 2^r$ for integer r = 1,2,3,…. Using the above construction, we obtain $D(c) = D(2^r) = \frac{3}{4}(r+1) = \frac{3}{4}(\log_2 c + 1)$. By slightly increasing this bound to $D(c) = \frac{3}{4}\left(\log_2\left(c+\frac{1}{2}\right)+1\right)$, we obtain a bound that holds for any integer $c \geq 0$; in fact this bound has the desirable properties that $D(0) = 0$ and $D(1)$ is not much larger than 1. Since *D(c)* is a concave positive function for all $c > 0$, an upper bound on the required server bandwidth (in units of the play rate) for arbitrary request rate λ_i is given by

$$B_{HMSM-optimal offline(2,1)} \leq \frac{3}{2}\left(\log_2\left(\frac{\lambda_i \mathrm{T_i}}{2}+\frac{1}{2}\right)+1\right) = \frac{3}{2\ln 2}\ln(\mathrm{N_i}+1).$$