Minimizing Bandwidth Requirements for On-Demand Data Delivery

Derek Eager
Mary Vernon
John Zahorjan

Technical Report #1418

March 2001 (Revised)

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.

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.

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 ($N_i = \lambda_i T_i$)
R_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.

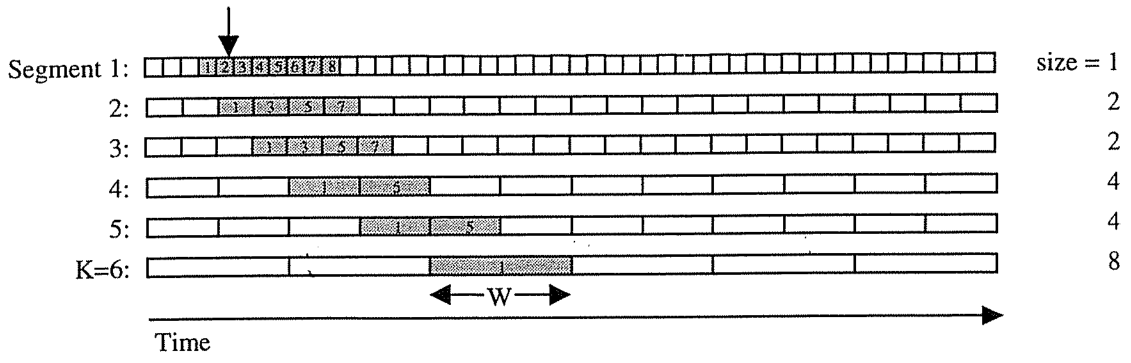

Figure 1: Skyscraper and Dynamic Skyscraper Delivery
(K=6, W=8, segment size progression = 1,2,2,4,4,8)

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.

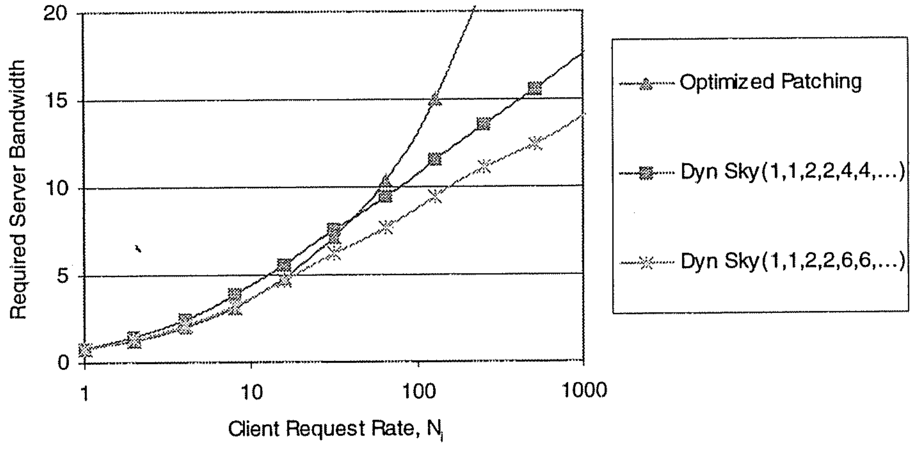

Figure 3: Required Server Bandwidth for Dynamic Skyscraper & Stream Tapping/Patching

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

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

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

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

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,....)

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) techniqueachieves 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 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:

$$R_{\mathrm{SSLRE}(n,1)} \approx \eta_{n,1} \ln\left(\frac{N_i}{\eta_{n,1}}+1\right), \qquad (6)$$

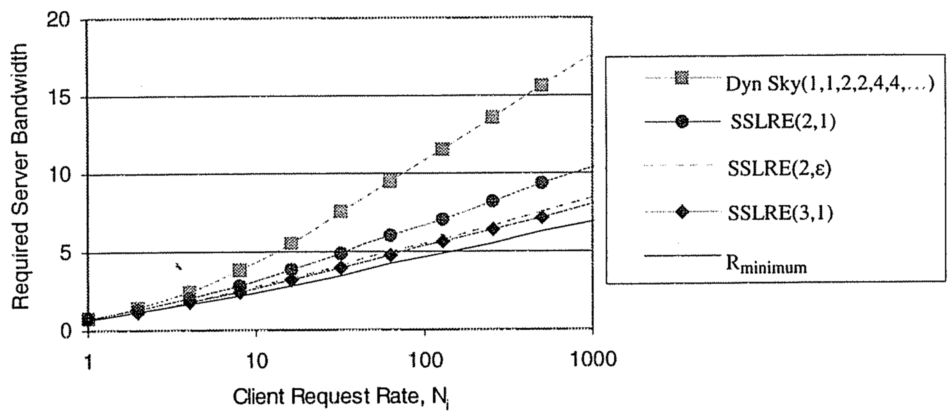

Figure 4: Required Server Bandwidth for Immediate Real Time File Delivery

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 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

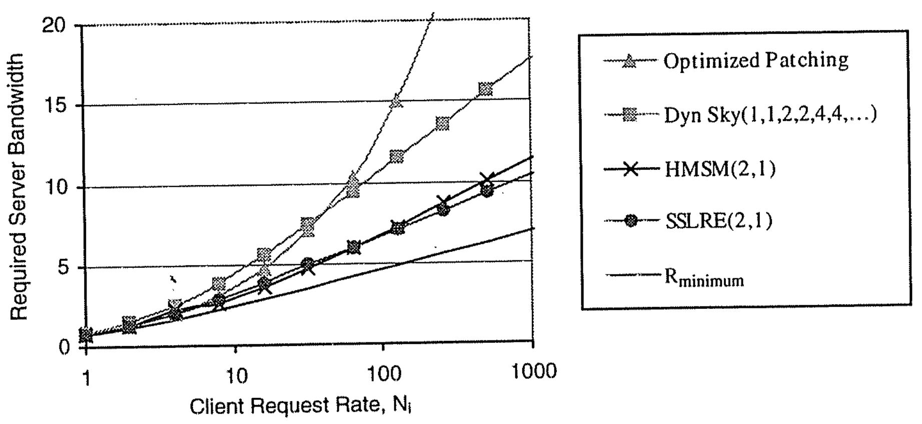

Figure 6: Required Server Bandwidth for Hierarchical Multicast Stream Merging

implement, assuming that $R_{SSLRE(n,1)}$ 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 $N_i \le 1000$, recalling from equation (6) that $R_{SSLRE(2,1)}$ 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:

$$R_{HMSM-optimaloffline(2,1)} \le 2.1\ln(N_i + 1)\,.$$

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 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.

[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.

[PaFl95] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling", *IEEE/ACM Transactions on Networking 3*, 3 (June 1995), pp. 244-266.

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

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

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

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 $R_{\text{SSLRE}(n,1)}$

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

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

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

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

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 2nd 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 R$_{\mathrm{SSLRE(n,r)}}$

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

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

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:

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

[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.

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 $T_i/2$ 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 $[D(c)/(T_i/2)]/(1/T_i) = 2D(c)$, where $c = \lambda_i T_i/2$.

Consider a cohort of integer size c. Clearly, we can choose $D(1)=1$. For $c \geq 2$, 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 $(T_i/2)^-$ after the request of the first client. Thus, in the following, we consider only this case.

For $c \geq 3$, 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 $[t, t+T_i/4)$ are merged together prior to merging with the rest of the cohort. Correspondingly, all those clients whose requests are made in the interval $[t+T_i/4, t+T_i/2)$ 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 $t+T_i/4$. (If there is at least one client, other than the latest client, that makes a request in the interval $[t+T_i/4, t+T_i/2)$, 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 $[t, t+T_i/4)$ occurred instead at time $t+T_i/4$, with an associated transmission cost of 1/2 of a full-file multicast.) Thus, in the following, we consider only this case.

For $c \geq 4$, and with the assumption that there is a client that made a request at time $t+T_i/4$, we refine our merging policy as follows. All those clients whose requests are made in the interval $[t+T_i/4, t+3T_i/8)$ are merged prior to merging with the rest of the cohort. Correspondingly, all those clients whose requests are made in the interval $[t+3T_i/8, t+T_i/2)$ 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 $t+(T_i/4)^-$. (If there is at least one client that makes a request in the interval $[t, t+T_i/4)$, 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 client in the interval $(t+T_i/4, t+T_i/2)$ occurred instead at time $t+(T_i/4)^-$ 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 $[t, t+T_i/4)$, and on the sub-cohort consisting of the arrivals in the (identically-sized) interval $[t+T_i/4, t+T_i/2)$. 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

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

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

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