

# Minimizing Bandwidth Requirements for On-Demand Data Delivery\*

**Derek Eager**

Dept. of Computer Science  
Univ. of Saskatchewan  
[eager@cs.usask.ca](mailto:eager@cs.usask.ca)

**Mary Vernon**

Computer Sciences Dept.  
Univ. of Wisconsin  
[vernon@cs.wisc.edu](mailto:vernon@cs.wisc.edu)

**John Zahorjan**

Dept. of Computer Science  
Univ. of Washington  
[zahorjan@cs.washington.edu](mailto:zahorjan@cs.washington.edu)

## Abstract

*Recent techniques for multicast or broadcast delivery of streaming media can provide immediate service to each client request yet achieve considerable client stream sharing (i.e., server and network bandwidth savings). This paper considers (1) the maximum savings in the required server (disk I/O and network) bandwidth that any such technique can provide, (2) the interplay between achievable reductions in required server bandwidth and available client receive bandwidth, (3) how well the previously proposed techniques perform relative to each other and to the minimum required server bandwidth for the assumed client capabilities, and (4) whether there are new practical delivery techniques that can achieve better server bandwidth savings than the previous techniques, yet still provide immediate service to client requests.*

*The principal results are as follows. First, we derive the minimum required server bandwidth for any delivery technique that provides immediate service to client requests, and find that this bandwidth grows logarithmically with the client request arrival rate. Second, we show that the minimum required server bandwidth can be nearly achieved if clients have receive bandwidth equal to three times the streaming rate and have sufficient storage for buffering data from shared streams. Third, we show that a particular implementation of the recently proposed partitioned dynamic skyscraper delivery technique provides immediate service to client requests more simply and directly than the original dynamic skyscraper method. Fourth, the recently proposed optimized stream tapping/grace patching/controlled multicast technique achieves nearly the minimum server bandwidth requirement at low client request rates, but the partitioned dynamic skyscraper delivery method has significantly lower server bandwidth requirements than optimized grace patching at moderate to high client request rates. Furthermore, the dynamic skyscraper technique has required server bandwidth within a small constant factor of the minimum required bandwidth, assuming sufficient client buffering capability. Finally, we propose a new practical delivery technique, namely hierarchical multicast stream merging, that has required server bandwidth significantly lower than for optimized grace patching or partitioned dynamic skyscraper, and close to the minimum achievable when client receive bandwidth is twice the streaming rate.*

## 1 Introduction

This paper considers the server (disk I/O and network) bandwidth required for true on-demand delivery of streaming media, such as video and audio files<sup>1</sup>. 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 large, 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 file 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<sup>2</sup>. 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 adjust display rates (i.e., for video files) so as to bring different streams to the same file position, at which time the streams can be merged [GoLM95, AgWY96, LaLG98]. An appealing aspect of these approaches is that they require the minimum possible client receive bandwidth (i.e., the streaming rate) and minimal client buffer space. On the other hand, if clients have receive bandwidth greater than the streaming rate, and some additional buffer space, significantly greater server bandwidth savings can be achieved [ViIm96, CaLo97, HuSh97, HuCS98, EaVe98, GaTo99, EaFV99, CaHV99, PaCL99, SGRT99]. In this case, clients receiving a particular multicast stream can

---

\* This work was supported in part by the NSF (Grants CCR-9704503 and CCR-9975044) and NSERC (Grant OGP-0000264).

<sup>1</sup> More generally, streaming may be fruitful for any file that clients will process incrementally and sequentially.

<sup>2</sup> We use the term “multicast” to denote both multicast and true broadcast throughout this paper.

*simultaneously* receive and buffer another portion of the data from a different stream, thus enabling greater opportunities for one client to catch up with and share future streams with another client.

A number of techniques have been proposed for reducing server bandwidth by having clients receive two or more data streams simultaneously. Two of these techniques, namely dynamic skyscraper (with channel stealing) [EaVe98] and stream tapping/patching/controlled multicast [CaLo97, HuCS98, CaHV99, GaTo99, SGRT99], have the property that they can provide *immediate* service to client requests. 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 maximum achievable savings in server (disk and network I/O) bandwidth for delivery techniques that provide such immediate service to clients?
- (2) What is the interplay between achievable server bandwidth reduction and available 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 client requests?

The principal system design results are as follows:

- In Section 2 we derive a tight lower bound on the required server bandwidth for any technique that provides immediate service to client requests. This lower bound grows *logarithmically* with the client request arrival rate, whereas if immediate service is provided and multicast delivery is not employed, required server bandwidth grows *linearly* with the request arrival rate.
- In Section 3 we show that the lower bound on the required server bandwidth for immediate service can be very nearly achieved if clients have receive bandwidth equal to just three times the streaming rate and clients can buffer the required data from shared streams.
- In Section 5.1 we show that a particular implementation of the *partitioned* dynamic skyscraper technique [EaFV99] provides immediate service to client requests more simply and directly than the original dynamic skyscraper method.
- In Section 5.2 we show that the dynamic skyscraper technique that requires client receive bandwidth equal to twice the streaming rate has required server bandwidth that grows *logarithmically* with the client request rate. Thus, at moderate to high client request rate, the dynamic skyscraper technique significantly outperforms the optimized stream tapping/grace patching/controlled multicast scheme, for which required server bandwidth grows with the *square root* of the request arrival rate (as shown in Section 4 and in [GaTo99]). However, there is still a significant gap between the bandwidth required for dynamic skyscraper delivery and the lower bound.
- In Section 6, a new delivery technique, *hierarchical multicast stream merging*, is proposed. Simulation results show that the average required server bandwidth for this new technique is close to the minimum achievable required server bandwidth when client receive bandwidth is twice the streaming rate.

For the purposes of obtaining the lower bound and examining the fundamental capabilities of the delivery techniques, the above results are obtained assuming that clients have sufficient space available for buffering the data streams, and that the entire file is consumed sequentially by the client, without use of VCR functions such as rewind or fast forward. However, each of the techniques that we consider can be adapted for limited client buffer space, and for VCR functions, with a concomitant increase in required server bandwidth. For example, limited buffering capability is examined in [EaVZ99b].

Our definition of *required* server bandwidth is the *average* server bandwidth used to satisfy client requests for a particular file, as a function of client request rate, when server bandwidth is unlimited; i.e., each client request is satisfied immediately. There are at least two reasons for believing that this is a good metric of the server bandwidth needed for a given client workload. First, although the server bandwidth consumed for delivery of a given file will vary over time, the variance over time in the *sum* of the bandwidths consumed for a reasonably large number of files is likely to be quite small, for a given total request rate. Thus, summing the average server bandwidths consumed for delivery of all files should yield an accurate estimate of the necessary total server capacity. Second, simulations of various delivery techniques show that when the finite server bandwidth capacity is equal to the sum of the average server bandwidths required for each file, average client waiting time is close to zero (e.g., [EaFV99, EaVZ99b]). If server bandwidth is reduced below this value, however, client delays or balking probabilities rapidly increase.

## 2 Lower Bound on Required Server Bandwidth for Immediate Service

Techniques that provide immediate service must initially allocate a separate stream in response to each new client request, so as to immediately begin delivery of the beginning portion of the requested file. Achieving bandwidth reductions through multicast delivery requires mechanisms that permit clients to share delivery of the later portion(s) of the file.

In this section, we derive a simple, yet tight, lower bound on the required server bandwidth as a function of client request rate, for any delivery technique that provides immediate service to client requests. The derivation assumes a Poisson arrival stream of requests. [EaVZ99a] shows that a similar result holds for a much broader class of arrival processes.

Let  $T_i$  be the duration of file  $i$  and  $\lambda_i$  be its average request rate. Consider an infinitesimally small portion of the file that begins at a position  $x$ , where position is measured by the elapsed time from the beginning when the file is processed at the streaming rate. For a system providing immediate service and guaranteeing delivery at least at the streaming rate, a multicast of this portion at some time  $t$  can be usefully received by those clients whose requests arrive in the interval  $[t-x, t]$ . That is, *at most* the clients that arrive within a window of duration  $x$  can share in a single multicast of this portion of the file. With Poisson arrivals, the average time from the end of one of these “catch-up windows” until the next request for file  $i$  is  $1/\lambda_i$ . Thus, the frequency of multicasts of the portion beginning at position  $x$  that would be required to support immediate service must be at least  $1/(x + 1/\lambda_i)$ . This yields the following lower bound on the required server bandwidth, in units of the streaming rate, for any technique that provides immediate service to client requests:

$$\int_0^{T_i} \frac{dx}{x + \frac{1}{\lambda_i}} = \ln(1 + T_i \lambda_i) = \ln(1 + N_i) \quad (1)$$

where  $N_i = \lambda_i T_i$  is the average number of requests for the file that arrive during a period of length  $T_i$ .

The above lower bound implies that, for Poisson arrivals, the required server bandwidth must grow at least logarithmically with the client request arrival rate. In fact, this is true for any particular form of arrival process such that the expected time until the next arrival, regardless of the time since the previous arrival, is bounded from above by  $c/\lambda_i$  for some constant  $c$  [EaVZ99a].

If each client has unlimited receive bandwidth and storage for buffering all data required for stream sharing, the lower bound in equation (1) is tight, as shown in the next section.

### 3 Impact of Limited Client Receive Bandwidth

For clients that have receive bandwidth equal to  $n$  times the streaming rate, for some integer  $n$ , we consider a delivery technique in which the file is divided into arbitrarily small segments and the following two rules are used to deliver the data 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 the reception of that multicast would not violate the limit  $n$  on the maximum receive bandwidth. If at some point in time there are more than  $n$  concurrent multicasts of different segments that the client could fruitfully receive, the client preferentially chooses to receive 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 transmission by the server at the streaming rate and at the latest possible time; i.e., if it begins at position  $x$ , it is scheduled for time  $t+x$ .

Note that this delivery technique, with  $n$  equal to infinity, achieves the lower bound on server bandwidth to any desired precision (by dividing the file into sufficiently small segments). This “optimal” delivery technique may be impractical, even if the assumptions about client capabilities were satisfied, as it results in very fragmented and complex delivery schedules. However, it 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.

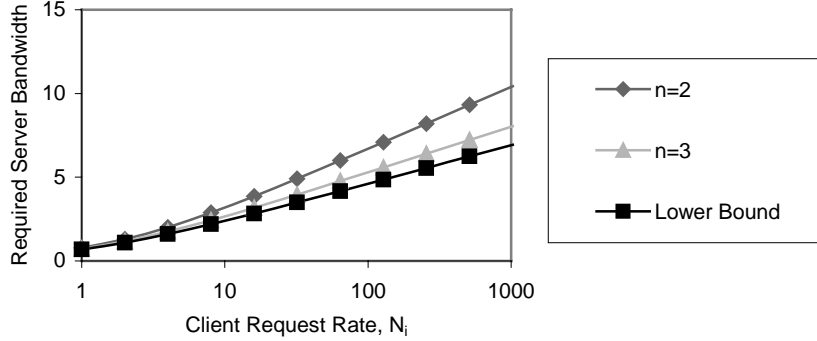
Determining a tight lower bound on the required server bandwidth when clients have limited receive bandwidth is complicated by issues concerning the optimal rearrangement of scheduled multicast transmissions when a new client request arrives. However, we speculate that the required server bandwidth for the above technique provides a good upper bound on the minimum required server bandwidth for each value of client receive bandwidth,  $n$ . We show in [EaVZ99a] that for a division of file  $i$  into sufficiently many small segments, the server bandwidth consumed using this delivery technique (in units of the streaming rate), for client receive bandwidth  $n$  and Poisson arrivals, is upper bounded by:

$$\eta_n \ln \left( 1 + \frac{N_i}{\eta_n} \right)$$

where  $\eta_n$  is the positive real constant that satisfies the following equation :

$$\eta_n \left( 1 - \left( \frac{\eta_n}{\eta_n + 1} \right)^n \right) = 1$$

Figure 1 shows the lower bound on the required server bandwidth for unlimited client receive bandwidth from equation (1), together with the upper bounds on the minimum required server bandwidth for  $n=2$  and  $n=3$  that are provided in this section, as functions of  $N_i$ . Note that  $\eta_n$  varies monotonically in  $n$  between  $\eta_2 = (1 + \sqrt{5})/2 \cong 1.62$  and  $\eta_\infty = 1$ . Thus, it may be possible to devise (practical) delivery techniques for client bandwidth capacity corresponding to  $n=2$ , that require no more than 62% greater server capacity than is minimally required when clients have unbounded receive bandwidth. Further, since  $\eta_3 \cong 1.19$ , nearly all of the benefit of unbounded client bandwidth, with respect to minimizing required server bandwidth, may be achievable when clients have receive bandwidth of only three times the streaming rate.



**Figure 1: Bounds on Required Server Bandwidth for Delivering a Given File**

(Client request rate,  $N_i$  equals the average number of requests that arrive during the time to stream the file)

#### 4 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]. The best of the proposed patching policies, *grace patching*, as well as a recently proposed variant that provides improvements in the case of constrained buffer space [SGRT99], is identical to the stream tapping policy if client buffer space is sufficiently large, as assumed for the purpose of 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 policy operates as follows. In response to some client requests, the server delivers the requested file in its entirety as 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 initial data that was delivered in the multicast stream prior to the client's request. Note that the required client receive bandwidth is thus twice the streaming rate. The patch stream terminates when it reaches the data already buffered by reception of the full-file multicast.

To keep unicast patch streams short, if 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. We let  $x_i$  denote the value of this threshold for file  $i$ . Assuming a Poisson request arrival stream, the required server bandwidth for delivery of file  $i$  using grace patching, measured in units of the streaming rate, is as follows:

$$\frac{T_i + \lambda_i x_i T_i \frac{x_i T_i}{2}}{x_i T_i + \frac{1}{\lambda_i}} = \frac{1 + \frac{x_i^2 N_i}{2}}{x_i + \frac{1}{N_i}}$$

The denominator is the average time that elapses between successive full-file multicasts. The numerator is the expected value of the sum of the transmission times of the full-file and patch streams in that interval.

Differentiating the above expression with respect to  $x_i$ , and setting the result to zero, as in [GaTo99] we obtain  $(\sqrt{2N_i + 1} - 1)/N_i$  as the optimal threshold value. Substitution into the above expression for server bandwidth yields:

$$\text{Required server bandwidth for optimized stream tapping/grace patching} = \sqrt{2N_i + 1} - 1$$

The above derivation of the optimal tapping/patching threshold and the required server bandwidth is very similar to, but somewhat simpler than the derivation in [GaTo99]. 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.

## 5 Required Server Bandwidth for Dynamic Skyscraper

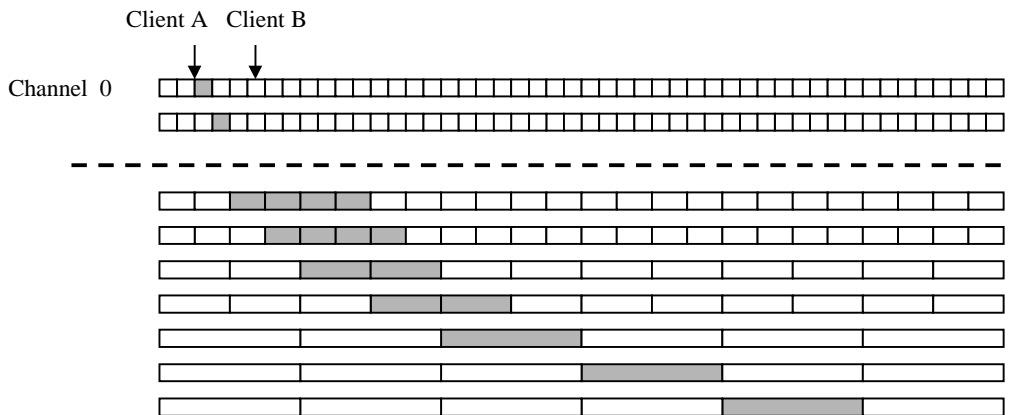
### 5.1 Providing Immediate Service Using Partitioned Dynamic Skyscraper

The dynamic skyscraper delivery technique, defined in [EaVe98], is based on the static skyscraper broadcast scheme defined in [HuSh97]. The technique is most easily described if all of the files have the same total size and streaming rate. In that case, total server bandwidth,  $C$ , expressed in units of the streaming rate, is conceptually organized into  $C/K$  groups of  $K$  “channels”, where each channel delivers data at the streaming rate. Files are divided into  $K$  segments of increasing size. In response to client requests for a file, a sequence of transmission periods is allocated on each of the channels within some particular group (forming a “transmission cluster”), and each segment  $j$  is repeatedly multicast on the  $j^{\text{th}}$  channel of the group. Since multiple multicasts of a smaller (i.e., earlier) segment are scheduled for each multicast of a larger segment, clients that receive different small segment multicasts may merge together and share multicasts of larger segments.

As shown in [EaVe98], a technique termed *channel stealing* can be used to effectively provide immediate service. Normally, all of the transmission periods within a transmission cluster are used to repeatedly multicast the segments of one particular file. With channel stealing, however, transmission periods that have no receiving clients may be reallocated to provide quick service to newly arriving requests.

We propose here a simple and more direct way to provide immediate service, using a particular application of *partitioned* skyscraper delivery [EaFV99], and a small modification to the segment size progression. The progression of relative segment sizes in the dynamic skyscraper technique, as originally proposed, has the form  $1, i, i, j, j, k, k, \dots$ , where each size increase can either be two-fold or three-fold. The progression is upper-bounded by a parameter,  $W$ , the purpose of which is to bound the amount of client buffer space required. Our proposed scheme for enabling immediate service entails adding an extra unit-segment at the start of the progression, thus creating the progression  $1, 1, i, i, j, j, k, k, \dots$ . We also partition the channels so that delivery of the first two unit segments can be scheduled immediately in response to a client request.

This partitioned skyscraper architecture, depicted in Figure 2, operates in the same way as the more general partitioned skyscraper system proposed in [EaFV99]. That is, the server bandwidth is logically organized into groups of two channels and groups of  $K-2$  channels, where one group of each type is shown in the figure. When client A requests file  $i$ , the server allocates bandwidth equal to a single transmission period (i.e., a degenerate transmission cluster) on channels 0 and 1, as well as a transmission cluster on channels 2 through  $K-1$ . These clusters are highlighted in the figure. When client B requests the file, the server only allocates a new degenerate cluster on channels 0 and 1 (possibly in a different group of two channels than the group that serves client A), which is used to deliver the first two segments of the file to client B. Client B also receives segments from the same transmission cluster on channels 2 through  $K-1$  that was allocated for client A.



**Figure 2: Partitioned Dynamic Skyscraper Delivery**

( $K=8$ ,  $W=8$ ; Segment Sizes:  $1, 1, 2, 2, 4, 4, 8, 8, 8$ )

A key observation is that this partitioned dynamic skyscraper system with progression  $1, 1, 2, 2, 4, 4, 8, 8, \dots$  requires client receive bandwidth equal to only twice the streaming rate (i.e.,  $n=2$ ), as in the skyscraper system without partitioning and the same segment size progression [EaVe98]. As well, the progression  $1, 1, 2, 2, 6, 6, 12, 12, 36, 36, \dots$  requires a client receive bandwidth of three times the streaming rate (as in the corresponding skyscraper system without partitioning).

## 5.2 Required Server Bandwidth

Let  $U$  denote the duration of a unit-segment multicast, which is determined by the time duration of the entire file ( $T_1$ ) and the segment size progression. For the immediate-service dynamic skyscraper system defined above, the required server bandwidth (measured in units of the streaming rate) for delivery of a file  $i$ , given a Poisson arrival stream of requests, is equal to  $2U\lambda_i + \frac{(K-2)WU}{WU + 1/\lambda_i}$ . The first term is the required bandwidth for delivering the first two unit segments of the file,

whereas the second term is the required bandwidth for delivering the transmission clusters (which have duration  $WU$  on each of the  $K-2$  channels) for the rest of the file.

The values of  $K$  and  $W$  that minimize the required server bandwidth, as given by the above expression, may be found numerically for any particular segment size progression of interest. It is also possible to determine the asymptotic behavior under high client request arrival rates. In this case, for any particular value of  $K$ , server bandwidth is minimized for maximal  $W$ . Thus, for the progression  $1,1,2,2,4,4,8,8,\dots$ , for  $K$  even, the optimal value of  $W$  is  $2^{(K/2)-1}$ , while for  $K$  odd, the optimal value of  $W$  is  $2^{(K-1)/2}$ . Furthermore, the optimal value of  $K$  grows without bound as the client request arrival rate grows without bound. Thus, for this segment size progression and with optimal  $W$ , asymptotically  $U$  is approximately  $T_1 / 4W$  for  $K$  even, and  $T_1 / 3W$  for  $K$  odd. Substituting the optimal values of  $U$  and  $W$  into the previous expression for required server bandwidth consumed and taking the derivative with respect to  $K$  yields expressions for the optimal even and odd values of  $K$ , which may then be used to find the required server bandwidth. Asymptotically, odd  $K$  minimizes server bandwidth usage, but in both cases the optimal  $K$  and the required server bandwidth are on the order of:

$$\frac{2}{\ln 2} \ln(N_i)$$

Note that the required server bandwidth for this partitioned skyscraper system grows only *logarithmically* with client request rate, and is within a small constant factor of the lower bound.

Other skyscraper systems may be similarly analyzed. For the progression  $1,1,2,2,6,6,12,12,36,36,\dots$ , both the optimal  $K$  and the minimal average required server bandwidth can be shown to be asymptotically on the order of:

$$\frac{4}{\ln 6} \ln(N_i)$$

Figure 4 considers the case of client receive bandwidth equal to twice the streaming rate, and plots the required server bandwidth for the skyscraper system with progression  $1,1,2,2,4,4,8,\dots$  and optimal choices of  $K$  and  $W$ , in comparison to the required server bandwidth for optimized grace patching and that for hierarchical multicast stream merging (Section 6), and the  $n=2$  bound from Section 3 (assuming Poisson request arrivals). Note that at low client request rates optimized grace patching has moderately lower server bandwidth requirements than the skyscraper system, while at high client request rates the skyscraper system has greatly reduced bandwidth requirements.

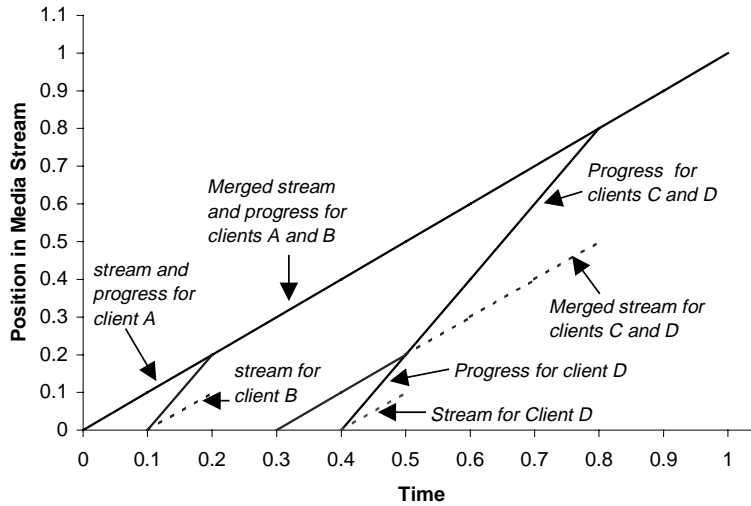
## 6 Hierarchical Multicast Stream Merging

Comparing the bandwidth requirements of optimized grace patching and dynamic skyscraper to our bounds, it appears that there is scope for improving performance through new techniques. Motivated by this result, in this section we propose a new approach, which we call *hierarchical multicast stream merging*. This new approach attempts to capture the advantages of dynamic skyscraper [EaVe98] and piggybacking [GoLM95, AgWY96, LaLG98], as well as the strengths of stream tapping/patching [CaLo97, HuCS98].

There are two essential elements of the hierarchical multicast stream merging approach. First, clients that request the same file repeatedly merge into larger and larger groups, leading to a hierarchical merging structure (as in dynamic skyscraper or piggybacking). Second, clients are merged using patch streams (as in stream tapping/patching) rather than using transmission clusters or adjusting playback rates.

The hierarchical multicast stream merging approach is illustrated in Figure 3 for a given set of request arrivals for a single file, assuming clients have receive bandwidth equal to twice the streaming rate. (In this case, 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 patch (or full-file) stream, as well as one's own stream.) 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. 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.

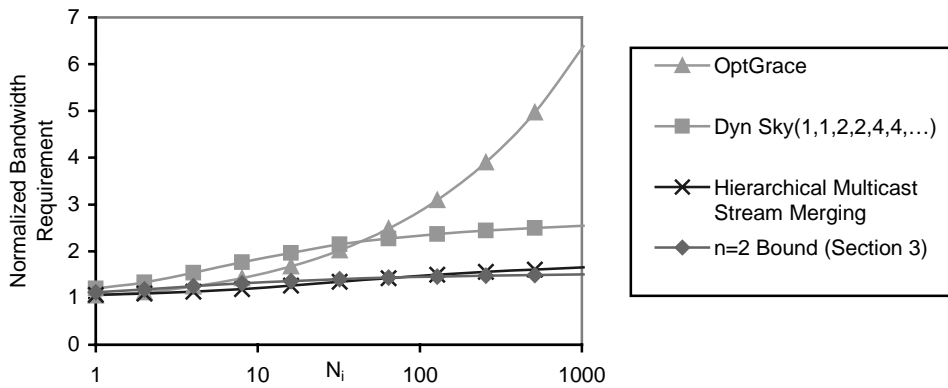
The dashed lines in the figure represent patch streams; the solid lines, which may hide portions of the patch streams, show the amount of data that a client or group of clients has accumulated as a function of time.



**Figure 3: Example of Hierarchical Multicast Stream Merging**

In the figure, requests are initiated by clients A, B, C, and D at times 0, 0.1, 0.3, and 0.4, respectively. Each new client request initiates a new transmission stream. 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 that stream before client C can merge with the stream initiated by client A. When C and D merge, both clients listen to the streams initiated by A and C until both clients have accumulated enough data to merge with clients A and B.

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 additional (existing or new) streams are listened to by clients so as to accomplish the desired merges. Some of these variants are investigated in [EaVZ99b] for homogeneous clients with receive bandwidth equal to twice the streaming rate; on-going research considers other contexts.



**Figure 4: Required Server Bandwidth for Hierarchical Multicast Stream Merging**

(Bandwidths are normalized by the lower bound on required bandwidth from equation (1),  $n=\infty$ )

Figure 4 provides the required server bandwidth for hierarchical multicast stream merging, as obtained from simulation, assuming client receive bandwidth equal to twice the streaming rate, Poisson arrivals, and optimal merges that are computed from known client request arrival times using a dynamic programming technique [AgWY96, EaVZ99b]. As shown in [EaVZ99b], there are simple heuristics for determining merges with unknown future client request arrival times that yield very nearly the same performance as for the offline optimal merges considered here. Figure 4 shows that

hierarchical multicast stream merging yields uniformly good performance, substantially improving on previous techniques. Further, comparison with the  $n=2$  bound from Section 3 suggests that there is little scope for further improvement, assuming that this bound provides accurate insight into the limits on achievable performance, which seems likely to be the case.

## 7 Conclusions

We have derived the minimum required server (disk I/O and network) bandwidth for any delivery technique that provides immediate service to client requests. Relative to this yardstick, we have evaluated previous techniques, and found substantial scope for improvement. This has motivated our proposed new delivery technique, hierarchical multicast stream merging, which has close to the minimum achievable required server bandwidth when client receive bandwidth is twice the streaming rate. Current research includes: (1) the design of hierarchical multicast stream merging policies for other client receive bandwidths and heterogeneous clients, (2) extending the technique for functions such as fast forward, indexing, and proxy caching, (3) investigating improved support for variable bit rate streams, and (4) the design and implementation of a prototype multimedia server that supports experimental evaluation of alternative delivery architectures.

## References

- [AgWY96] 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.
- [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. 6<sup>th</sup> 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. 2<sup>nd</sup> 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. 4<sup>th</sup> 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.
- [EaVZ99a] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Minimizing Bandwidth Requirements for On-Demand Data Delivery", Tech. Report #4105, Computer Sciences Dept., University of Wisconsin – Madison, Aug. 1999.
- [EaVZ99b] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Optimal and Efficient Merging Schedules for Video-on-Demand Servers", *Proc. 7<sup>th</sup> ACM Int'l. Multimedia Conf. (ACM MULTIMEDIA '99)*, Orlando, FL, Nov. 1999.
- [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 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. 6<sup>th</sup> ACM Int'l. Multimedia Conf. (ACM MULTIMEDIA '98)*, Bristol, U.K., Sept. 1998, pp. 191-200.
- [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.
- [SGRT99] S. Sen, L. Gao, J. Rexford, and D. Towsley, "Optimal Patching Schemes for Efficient Multimedia Streaming", *Proc. 9<sup>th</sup> 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.