SIAM J. COMPUT.
Vol. 33, No. 5, pp. 1011–1034

EFFICIENT ALGORITHMS FOR OPTIMAL STREAM MERGING FOR MEDIA-ON-DEMAND[*]

AMOTZ BAR-NOY[†] AND RICHARD E. LADNER[‡]

Abstract. We address the problem of designing optimal off-line algorithms that minimize the required bandwidth for media-on-demand systems that use stream merging. We concentrate on the case where clients can receive two media streams simultaneously and can buffer up to half of a full stream. We construct an $O(nm)$ optimal algorithm for n arbitrary time arrivals of clients, where m is the average number of arrivals in an interval of a stream length. We then show how to adopt our algorithm to be optimal even if clients have a limited size buffer. The complexity remains the same.

We also prove that using stream merging may reduce the required bandwidth by a factor of order $\rho L/\log(\rho L)$ compared to the simple batching solution where L is the length of a stream and $\rho \leq 1$ is the density in time of all the n arrivals. On the other hand, we show that the bandwidth required when clients can receive an unbounded number of streams simultaneously is always at least 1/2 the bandwidth required when clients are limited to receiving at most two streams.

Key words. media-on-demand, stream merging, dynamic programming, monotonicity property

AMS subject classifications. 68W05, 68W40

DOI. 10.1137/S0097539701389245

1. Introduction.

Media-on-demand is the demand by clients to play back, view, listen to, or read various types of media such as video, audio, and large files with as small as possible startup delays and with no interruptions. The solution of dedicating a private channel to each client for the required media is implausible even with the ever growing available network bandwidth. Thus, multicasting popular media to groups of clients seems to be the ultimate solution to the ever growing demand for media. The first, and most natural, way to exploit the advantage of multicasting is to *batch* clients together. This implies a trade-off between the overall server bandwidth and the guaranteed startup delay. The main advantage of the batching solutions lies in their simplicity. The main disadvantage is that the guaranteed startup delay may be too large.

The pyramid broadcasting paradigm, pioneered by Viswanathan and Imielinski [43, 44], was the first solution that dramatically reduced the bandwidth requirements for servers by using larger receiving bandwidth for clients and by adding buffers to clients. Many papers have followed this line of research; all of them have demonstrated the huge improvement over the traditional batching solutions. We adopt the *stream merging* technique, introduced by Eager, Vernon, and Zahorjan [18, 19]. Stream merging seems to incorporate all the advantages of the pyramid broadcasting paradigm and is very useful in designing and implementing efficient off-line and on-line solutions.

A system with stream merging capabilities is illustrated in Figure 1. The server multicasts the popular media in a staggered way via several channels. Clients may

[*]Received by the editors May 10, 2001; accepted for publication (in revised form) January 25, 2004; published electronically June 25, 2004.
http://www.siam.org/journals/sicomp/33-5/38924.html

[†]Computer and Information Science Department, Brooklyn College – CUNY, 2900 Bedford Avenue, Brooklyn, NY 11210 (amotz@sci.brooklyn.cuny.edu). This work was done in part while the author was a member of the AT&T Labs-Research, Shannon Lab, Florham Park, NJ.

[‡]Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195 (ladner@cs.washington.edu). This work was done in part at AT&T Labs-Research, Shannon Lab, Florham Park, NJ and was partially supported by NSF grants CCR-9732828 and CCR-0098012.

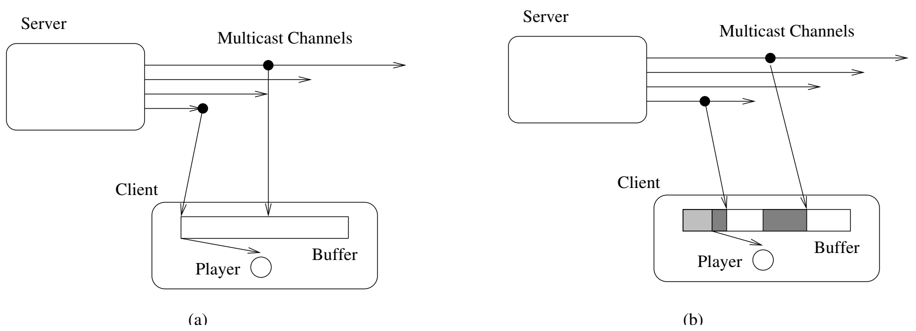

FIG. 1. *The mechanism of receiving data from two streams simultaneously.*

receive data from two streams simultaneously while playing data they have accumulated in their buffers. The playback rate is identical to each of the channels, so that the receiving bandwidth is twice the playback bandwidth. The initial position is illustrated in (a) where the client is about to receive data from a new stream and a stream that was initiated earlier. After some time the system may look as illustrated in (b). The client still receives data from both streams. The top of its buffer, which represents the beginning of the stream, has been viewed by the player. This technique is called stream merging because eventually, as the client receives both the earlier and later streams, it no longer needs the later stream because it already has the data from buffering the earlier one. At this point, if no other client needs the later stream, it can terminate. In a sense the later stream merges with the earlier one, forming just one stream. The termination of the later stream is where bandwidth is saved.

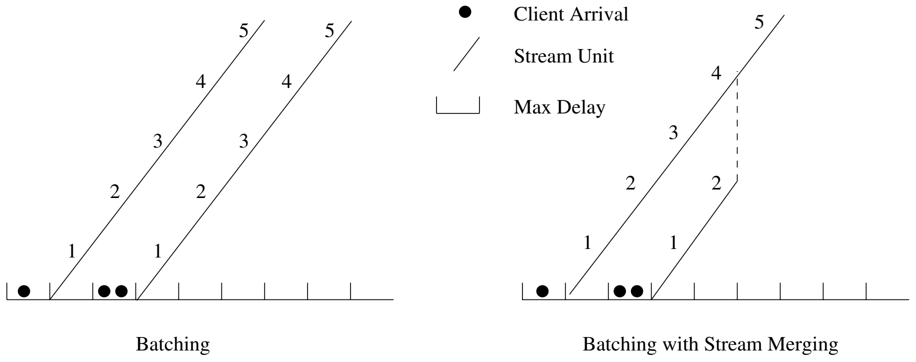

FIG. 2. *The figure on the left shows batching, while the figure on the right shows batching with stream merging.*

It is interesting to contrast stream merging with *batching*, the most common technique for reducing server bandwidth in the presence of multicast. In batching time is divided into intervals. A client that arrives in an interval is satisfied by a full stream at the end of the interval. Bandwidth is saved at the expense of longer guaranteed startup delay for the clients. Stream merging and batching can be combined so that there is a bandwidth saving from both stream merging and batching. Figure 2 shows the difference between pure batching and stream merging with batching. In this figure full streams are of length 5. The three clients require 2 full streams (10 units) with batching alone, but require only 1.4 streams (7 units) with stream merging. The second and the third clients receive parts 3 and 4 of the first stream at the same time they receive parts 1 and 2 from the second stream; then they receive part 5 from the first stream.

Given a sequence of arrivals, there can be a number of different stream merging solutions to accommodate this sequence. Typically, a stream merging solution has a number of full streams each associated with a number of other truncated streams that eventually merge to this full stream. We measure the bandwidth required by a solution as the sum of the lengths of all the individual (full or truncated) streams in the solutions. We call this sum the *full cost* of the solution. This cost represents the total bandwidth required by the solution and by dividing it by the time span of arrivals it represents the average bandwidth to serve the clients during that time span. In our example of Figure 2, the full cost of the batching solution is 10 units (or 2 streams) and the full cost of the stream merging with the batching solution is 7 units (or 1.4 streams).

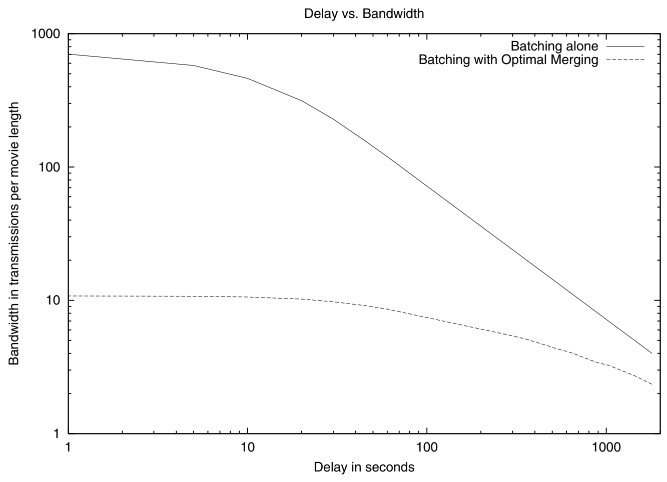

FIG. 3. *Comparison of bandwidth required for batching and batching with optimal stream merging. The figure plots the bandwidth requirement vs. delay for a* 2*-hour movie, with Poisson arrivals averaging every* 10 *seconds.*

Figure 3 shows the bandwidth requirement vs. delay for a popular 2-hour movie, with Poisson arrivals averaging every 10 seconds. The guaranteed startup delay ranges from 1 second to 30 minutes. For stream merging with batching we used an optimal stream merging algorithm. At 1 second delay the difference in bandwidth is dramatic. For batching the bandwidth required is almost the same as it would be if each client had its own stream. On the other hand, at 1 second delay, stream merging with batching uses 1/60 the bandwidth of batching.

1.1. Contributions. The main goal of this paper is to find efficient ways to compute the optimal stream merging solutions, those that minimize the full cost. To determine an optimal solution, we have to decide when to start full streams and how to merge the rest of the streams into the full streams. We assume that the arrival times of clients are known ahead of time and we call this the *off-line* problem, as opposed to the *on-line* problem where client arrivals are not known ahead of time. Computing the optimal off-line solution quickly is a major focus of this paper. The off-line scenario happens when clients make reservations for when their streams will begin. However, good on-line solutions are required for media-on-demand systems that run in real time. The optimal off-line solution is the gold standard against which on-line solutions should be compared. Fast algorithms for computing an optimal off-line solution allow us to evaluate the quality of on-line solutions on large numbers of media requests.

In our main model a client is capable of receiving two streams simultaneously. We call this the *receive-two* model. It is instructive to consider the *receive-all* model in which a client is capable of receiving any number of streams simultaneously. There are several reasons to consider this case. First, we will see that there is very little gain in going from the receive-two model to the receive-all model. Most of the benefit of stream merging comes from just the ability to receive two streams simultaneously. Second, we will see that many of the results from the receive-two model carry over in a simpler form to the receive-all model.

Our first contribution is a novel model for the stream merging technique. A key concept in our model is that of a *merge tree*, which is an abstraction of the diagram in Figure 2. See Figure 4 for an example. A sequence of merge trees is called a *merge forest*. The root of a merge tree represents a full stream; its structure represents the merging pattern of the remaining streams that are associated with its descendent. A sequence of merge trees is called a *merge forest*. We show that the knowledge of the arrival times and the structure of the merge trees is sufficient to compute the lengths of all the streams and to compute the full cost (total bandwidth) required by the merge forest. A key component of our approach is the concept of *merge cost*. For a given merge tree, the merge cost is the sum of the lengths of all the streams except the root stream. The full cost counts everything, merge cost and the length of the roots for all the merge trees in the forest. This separation into merge cost and full cost helps in designing the optimal algorithms and in having a cleaner analysis. Later in the paper, we first show how to construct an optimal merge tree for a sequence that forms a single tree and then show how to construct the optimal merge forest for a given sequence.

We show several properties that optimal merge trees must have. For example, there is no gain in having streams that do not start at an arrival time of some clients. Other properties will be defined in section 2. These properties were assumed implicitly by all the on-line algorithms that use the stream merging technique [18, 19, 5, 14, 11, 12]. Thus our model, in a way, builds the foundations for designing "good" on-line algorithms.

Our main focus is in designing efficient optimal algorithms in the receive-two model, that is, algorithms that, for a given a sequence of arrivals, either find a merge tree that minimizes merge cost or find a merge forest that minimizes full cost. We have the following results depending on n, the number of arrivals. For the merge cost, we present an efficient $O(n^2)$ time algorithm improving the known $O(n^3)$ time algorithm (see [2, 19]). The latter algorithm is based on a straightforward dynamic programming implementation. Our algorithm implements the dynamic programming utilizing the monotonicity property ([34]) of the recursive definition for the merge cost. For the full cost we use the optimal solution of the merge cost as a subroutine. We describe an $O(nm)$ time algorithm where m is the average number of arrivals in an interval that begins with an arrival and whose length is a full stream length. We also have efficient algorithms for a model in which clients have a limited buffer size. We maintain the $O(nm)$ complexity where this time m is the average number of arrivals in an interval that begins or ends with an arrival and whose length is the minimum between the stream length and the maximum buffer size.

Additional results establish the performance of the optimal stream merging solutions. Let L be the length of a full stream in slots, where a *slot* is the worst-case waiting time for any arrival before it receives the first segment of the media. For a fixed length media, as the parameter L grows the waiting time tends to zero. Define $\rho \leq 1$ to be the ratio of slots that have at least one arrival to all the slots in a given period of time. We show that an optimal stream merging solution reduces the required full cost by a factor of order $\rho L/\log(\rho L)$ for the full cost compared to the simple batching solution. Note that the improvement is huge for large L because simple batching solutions must dedicate a full stream for each arrival. However, L cannot grow forever because then ρ would approach zero.

Finally, we present optimal algorithms for the receive-all model that have the same time complexity bounds as the receive-two model. We show that the full cost required in an optimal solution in the receive-all model is always at least half the optimal full cost required in the receive-two model.

1.2. Related research. Several papers (e.g., [15, 13, 3]) proposed various batching solutions demonstrating the trade-off between the guaranteed startup delay and the required server bandwidth. The solutions are simple but may cause large startup delays. The seminal pyramid broadcasting solution [43, 44] was the first paper to explore the trade-off with two other resources: the receiving bandwidth of clients and the buffer size of clients. Many researchers were concerned about reducing the buffer size (see e.g., [1]). However, all of them demonstrated the huge improvement over the traditional batching solutions.

The skyscraper broadcasting paper [27] showed that the receive-two model already exploits the dramatic improvement. Researchers also demonstrated the trade-off between the server bandwidth and the receiving bandwidth [26, 20, 37, 38] in this framework. All of these papers assumed a static allocation of bandwidth per transmission. The need for dynamic allocation (or on-line algorithms) motivated the papers [17, 16] that still used the skyscraper broadcasting model. The patching solution [25, 21, 8], the tapping solution [9, 10], the piggybacking solution [2, 23, 24, 35], and the stream merging solution [18, 19] assumed the attractive dynamic allocation of bandwidth to transmissions. The early papers regarding patching assumed that clients may merge only to full streams. Later papers regarding patching assumed a model that is essentially the stream merging model. New research regarding patching [40] assumed that streams may be fragmented into many segments.

The original stream merging algorithms [18, 19] were on-line and event-driven, where telling clients which streams to listen to was done at the time of an event. The specific events were the arrival of a client, the merge time of two streams, and the termination of a stream. The papers reported good practical results compared to the optimal algorithm on Poisson arrivals. These event-driven algorithms are quite different in character from the series of on-line algorithms that appeared subsequently [5, 14, 11, 12]. Unlike in the event-driven algorithms, in the newer algorithms, a client learns all the streams it will be receiving from at the time it arrives. The dynamic Fibonacci tree algorithm of [5] used merge trees and had a competitive analysis. Next, the dyadic algorithm [14] was proposed and analyzed for its average performance on Poisson arrivals. Next, an algorithm based on a new version of merge trees called rectilinear trees was shown to be 5-competitive (full cost no more than 5 times that of the optimal) [11]. Later these same authors proved that the dyadic algorithm is 3-competitive [12]. A comparison of the performance of on-line stream merging algorithms can be found in [4].

Finally, the following is a partial list of additional papers that address trade-offs among the four parameters: server bandwidth, delay guaranteed, receiving bandwidth, and buffer size: [28, 29, 30, 31, 32, 36, 39, 7, 22, 41, 42].

1.3. Paper organization. In section 2 we define our stream merging model and prove properties of optimal solutions. Section 3 presents our algorithm in the receive-two model with unbounded buffers. In section 4, we consider the limited buffer size case. In section 5, we describe our results for the receive-all model. Finally, we discuss our results and some related problems in section 6.

2. The stream merging model.

Basic definitions. Assume that time starts at 0 and is slotted into unit sized intervals. For example, a 2-hour movie could be slotted into time intervals of 15 minutes. Thus, the movie is 8 units long. For a positive integer t, call the slot that starts at time $t-1$ slot t. The length of a full stream is L units. There are n arrival times for clients denoted by integers $0 \leq t_1 < t_2 < \cdots < t_n$. Clients that arrive at the same time slot are considered as one client. At each arrival time a stream must be scheduled, although for a given arrival the stream may not run until conclusion because only an initial segment of the stream is needed by the clients. A client may receive and buffer data from two streams at the same time while viewing the data it accumulated in its buffer. The objective of each client is to receive all the L parts of the stream and to view them without any interruption starting at the time of its arrival.

At this point we would like to note the following:

- We will show later that there is no gain in scheduling streams except at arrival times. Hence, it is very useful to use the client arrival time t as both a name for the client that arrives at time t and for the stream that is initiated at time t. Moreover, for ease of presentation, in the rest of this section we assume that only such streams exist.
- Our results hold for the nondiscrete time model as well by letting the time slots be as small as desired and therefore the value of L as large as needed. We adopt the discrete time model for ease of presentation.

Merge forests and merge trees. A *solution* to an arrival sequence is a *merge forest* which is a sequence of *merge trees*. A merge tree is an ordered labeled tree, where each node is labeled with an arrival time and the stream initiated at that time. The root is labeled t_1 and if a nonroot node is labeled t_i, then its parent is labeled t_j, where $j < i$. This requirement means that a stream can merge only to an earlier stream. Additionally, if t_j is a right sibling of t_i then $j > i$. This requirement means that the children of a node are ordered by their arrival times. Clearly, in a merge forest all the arrival times in one tree must precede the arrival times in the successive tree. We say that an ordered labeled tree has the *preorder traversal property* if a preorder traversal of the tree yields the arrival times in order. Any ordered labeled tree with the preorder traversal property is a merge tree, but not necessarily vice versa. We will see later in Lemma 2.2 that every optimal merge tree satisfies the preorder traversal property.

Figure 4 illustrates a merge tree and a concrete diagram showing how merging would proceed for the given merge tree. In the concrete diagram each arrival is shown on the time axis and for each arrival a new stream is initiated. The vertical axis shows the particular unit of the stream that is transmitted. The root stream, t_1, is of full length, while all the other streams are truncated. A stream is truncated because all

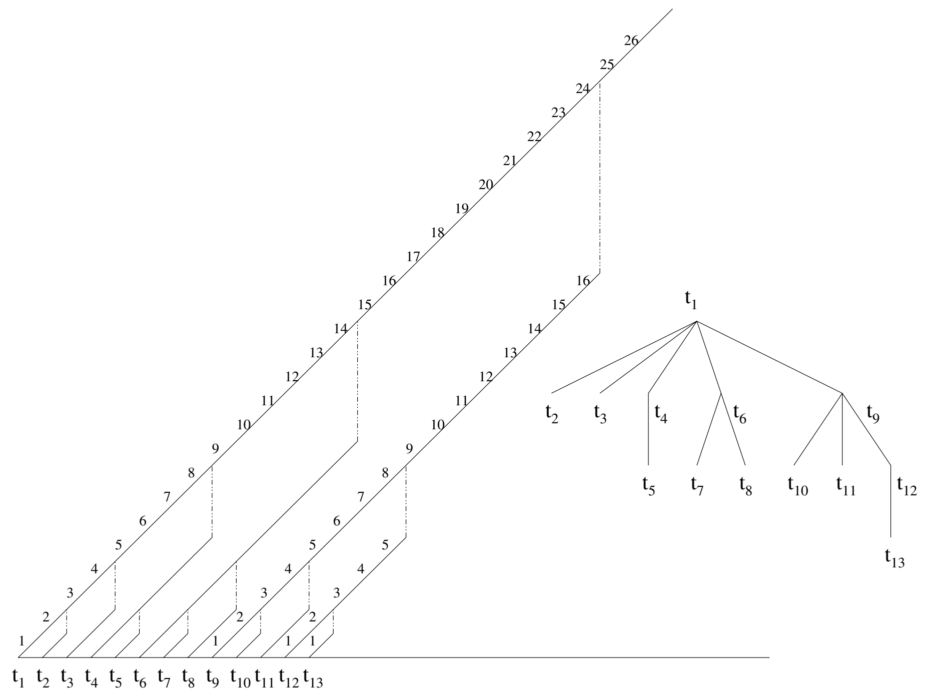

FIG. 4. *On the left is a concrete diagram showing the length of each stream with its merging pattern. On the right is its corresponding merge tree. In this example there are* 13 *arrivals at times* $0, \ldots, 12$.

the clients that were receiving the stream no longer need any data from it, having already received the data from some other stream(s). Note that although the merge tree does not show the stream lengths, it implicitly contains all the information in the concrete diagram, as will be shown in Lemma 2.1.

For the moment, we postpone the explanation of how we calculate the lengths of the truncated streams because we need more explanations of how merging works. Nonetheless, we can now explain that the problem we are addressing is how to find a solution that minimizes the sum of the lengths of all the streams in the solution. This is equivalent to minimizing the total number of units (total bandwidth) needed to serve all the clients. Minimizing the total bandwidth is essentially the same as minimizing the average bandwidth needed to satisfy the requests. The average bandwidth required to satisfy the requests by the forest F is the sum of the total bandwidth required by F divided by $(t_n - t_1)$ which is the time span of the n arrivals. The equivalence follows since the quantity $t_n - t_1$ is independent of the solution.

Receiving procedures. Clients receive and buffer data from various streams according to their location in the forest. At any one time a client can receive data from at most two streams. Informally, a client arriving at time x receives data from all the nodes on the path from x to the *root* of the tree. At the same time it receives data from a node y and its parent until it does not need any more data from the node. At that point the client moves closer to the root by receiving data from the parent of y and its parent. We call this transition a merge operation. In the following we formally define the actions of a client in the merge tree.

Let $x_0 < x_1 < \ldots < x_k$ be the path from the root x_0 to node x_k that is the arrival time of a specific client. We call this sequence of length $k+1$ the *receiving procedure* of the client. Denote by $x_0, x_1, \ldots, x_k$ the streams that are scheduled at the corresponding arrival times. The client obeys the following *stream merging rules*.

Stage i, $0 \le i \le k-1$: For $x_{k-i} - x_{k-i-1}$ time slots from time $2x_k - x_{k-i}$ to time $2x_k - x_{k-i-1}$ the client receives parts $2x_k - 2x_{k-i} + 1, \ldots, 2x_k - x_{k-i} - x_{k-i-1}$ from stream x_{k-i} and parts $2x_k - x_{k-i} - x_{k-i-1} + 1, \ldots, 2x_k - 2x_{k-i-1}$ from stream x_{k-i-1}.

Stage k: For $L - 2(x_k - x_0)$ time slots from time $2x_k - x_0$ to time $x_0 + L$ the client receives parts $2(x_k - x_0) + 1, \ldots, L$ from stream x_0.

This describes how the client arriving at x_k receives the entire transmission of the stream. In particular, part j of the stream is received in stage $i = k$ if $2(x_k - x_0) < j$ and in stage $i < k$ if $2(x_k - x_{k-i}) < j \le 2(x_k - x_{k-i-1})$. Notice that if $x_k - x_0 \le \lfloor L/2 \rfloor$, then the client is busy receiving data for $L - (x_k - x_0)$ time slots since in $x_k - x_0$ slots it receives data from two streams, and if $x_k - x_0 > \lfloor L/2 \rfloor$, then the client is busy receiving data for $x_k - x_0$ time slots since in $L - (x_k - x_0)$ slots it receives data from two streams.

Consider the example depicted in Figure 4. Assume a full stream of length 26 and the following stream merging rules for the client that arrives at time $t_{13} = 12$. In this case, we have $k = 3$ with $x_0 = 0, x_1 = 8, x_2 = 11, x_3 = 12$. From time 12 to time 13 the client receives part 1 from stream x_3 and part 2 from stream x_2. From time 13 to time 16 the client receives parts $3, \ldots, 5$ from stream x_2 and parts $6, \ldots, 8$ from stream x_1. From time 16 to time 24 the client receives parts $9, \ldots, 16$ from stream x_1 and parts $17, \ldots, 24$ from stream x_0. Finally, from time 24 to time 26 the client receives parts $25, 26$ from stream x_0.

Length of streams. Given the stream merging rules, we must still determine the minimum length of each stream so that all the clients requiring the stream receive their data. In a merge tree T the root is denoted by $r(T)$. If x is a node in the merge tree, then we define $\ell_T(x)$ to be its *length* in T. That is, $\ell_T(x)$ is the minimum length needed to guarantee that all the clients can receive their data from stream x using the stream merging rules. For a nonroot node x define $p_T(x)$ to be its parent and $z_T(x)$ to be the latest arrival time of a stream in the subtree rooted at x. If x is a leaf, then $z_T(x) = x$. We drop the subscript T when there is no ambiguity.

We can see from our definition of the stages that the length L of the root stream must satisfy $z - r(T) \le L - 1$, where z is the last arrival in the merge tree T. Otherwise, the clients arriving at z do not receive data from the stream initiated at $r(T)$. The next lemma shows how to compute the lengths of all the nonroot streams.

LEMMA 2.1. *Let $x \ne r(T)$ be a nonroot node in a tree T. Then*

$$\ell(x) = 2z(x) - x - p(x). \tag{1}$$

In particular, if x is a leaf, then $\ell(x) = x - p(x)$ since $z(x) = x$.

Proof. First observe that if clients $y' < y$ both receive data from x, then client y receives later parts of the stream x. This implies that the length of the stream x is dictated by the needs of the client that arrives at time $z(x)$. Let $x_0, x_1, \ldots, x_k$ be the path from the root of the tree T that contains both x and $z(x)$. That is, $x = x_i$ and $p(x) = x_{i-1}$ for some $i > 0$ and $z(x) = x_k$. By the stream merging rule of stage $k - i$, the client $z(x)$ receives data from the stream $x = x_i$ until time $2x_k - x_{i-1} = 2z(x) - p(x)$. Since $z(x)$ is the last client requiring stream x, then no more transmission of stream x is required. Since the stream x begins at time x and ends at time $2z(x) - p(x)$, its length is $2z(x) - x - p(x)$. ∎

In this paper, we will use for $\ell(x)$ either expression (1) or the following two alternative expressions:

$$\ell(x) = (x - p(x)) + 2(z(x) - x) \tag{2}$$
$$= (z(x) - x) + (z(x) - p(x)). \tag{3}$$

Expression (3) could be viewed as follows. The length of the stream x is composed of two components. The first component is the time needed for clients arriving at time x to receive data from stream x before they can merge with stream $p(x)$. The second component is the time stream x must spend until the clients arriving at time $z(x)$ merge to $p(x)$.

2.1. The merge cost. Let T be a merge tree. The *merge cost* of T is defined as

$$\text{Mcost}(T) = \sum_{x \neq r(T) \in T} \ell(x).$$

That is, the merge cost of a tree is the sum of all lengths in the tree except the length of the root of the tree. For an arrival sequence $t_1, \ldots, t_n$, define the *optimal merge cost* for the sequence to be the minimum cost of any merge tree for the sequence. An *optimal merge tree* is one that has optimal merge cost. The following technical lemma justifies restricting our attention to merge trees with the preorder traversal property.

LEMMA 2.2. *Every optimal merge tree satisfies the preorder traversal property.*

Proof. The proof is by induction on the number of arrivals. The lemma is clearly true for one arrival. Assume we have $n > 1$ arrivals and the lemma holds for any number of arrivals less than n. Let T be an optimal merge tree for the arrivals and let x be the last arrival to merge to the root r of T. Define T_R to be the subtree of T rooted at x and let T_L be the subtree of T obtained by removing T_R. By the induction hypothesis we can assume that T_R and T_L both have the preorder traversal property. Let w be the last arrival in the subtree T_L and let z be the last arrival in the subtree T_R. If $w < x$, then the entire tree T must already have the preorder property, and we are done. We need consider only the case where $w > x$. In this case we will construct another merge tree T' for the same arrivals whose merge cost is less than T's, contradicting the optimality of T.

Define a *high tree* to be a subtree of T_L whose root is greater than x and whose parent of the root is less than x. Let T' be the tree T where all the high trees are removed from T_L and are inserted as children of x. Naturally, the high trees must be inserted so that all the children of x in T' are in arrival order. For all nodes u in T such that $u \neq x$ or u is not an ancestor of a root of a high tree, we have $\ell_T(u) = \ell_{T'}(u)$. For an ancestor u of a root of a high tree we have $\ell_T(u) > \ell_{T'}(u)$, and for x we have $\ell_T(x) < \ell_{T'}(x)$. Let p be the parent of the root of the high tree containing w. We must have $p \neq r$ for otherwise x would not be the last arrival to merge to the root because the root of the high tree containing w is greater than x. We can just examine the change in length of the nodes p and x. We have

$$\text{Mcost}(T) - \text{Mcost}(T') \geq \ell_T(p) - \ell_{T'}(p) + \ell_T(x) - \ell_{T'}(x).$$

Let w' be the largest arrival in the tree rooted at p in T'. We must have $w' < x$; otherwise, w' is in some high tree that was removed from T_L and made a child of x in T'. Since w is the largest arrival in the tree rooted at p in T, we have $\ell_T(p) - \ell_{T'}(p) =$

$2(w - w')$ by Lemma 2.1. There are two cases to consider depending on whether $w < z$ or $w > z$. If $w < z$, then $\ell_T(x) = \ell_{T'}(x)$ because z is the largest arrival in the subtree rooted at x in both T and T'. By definition $w > w'$; hence,

$$\mathrm{Mcost}(T) - \mathrm{Mcost}(T') \geq 2(w - w') > 0.$$

If $w > z$, then $\ell_T(x) - \ell_{T'}(x) = 2(z - w)$ by Lemma 2.1. Hence,

$$\mathrm{Mcost}(T) - \mathrm{Mcost}(T') \geq 2(w - w') + 2(z - w) = 2(z - w') > 0$$

because $w' < x \leq z$. ∎

Lemma 2.2 allows us to consider only merge trees with the preorder traversal property. As a consequence, henceforth, *we assume that all merge trees have the preorder traversal property.* Hence, a key property of merge trees is that for any node t_i, the subtree rooted at t_i contains the interval of arrivals $t_i, t_{i+1}, \ldots, t_j$, where $z(t_i) = t_j$. Furthermore, t_j is the rightmost descendant of t_i. As a result, we can recursively decompose any merge tree into two in a natural way as shown in the following lemma and seen in Figure 5.

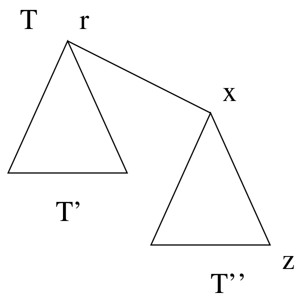

FIG. 5. *The recursive structure of a merge tree T with root r. The last arrival to merge directly with r is x. All the arrivals before x are in T' and all the arrivals after x are in T'' and z is the last arrival.*

LEMMA 2.3. *Let T be a merge tree with root r and last stream z and let x be the last stream to merge to the root of T.*

$$\mathrm{Mcost}(T) = \mathrm{Mcost}(T') + \mathrm{Mcost}(T'') + 2z - x - r, \tag{4}$$

where T' is the subtree of all arrivals before x including r and T'' is the subtree of all arrivals after and including x.

Proof. The length of any node in T' and T'' is the same as its length in T. Since the root of T' is the root of T, it follows that x is the only node in $\mathrm{Mcost}(T)$ whose length is not included in $\mathrm{Mcost}(T')$ or $\mathrm{Mcost}(T'')$. The lemma follows, since by Lemma 2.1 the length of x is $2z(x) - x - p(x) = 2z - x - r$. ∎

We now prove that there is no gain in broadcasting a prefix of the full stream if there is no arrival for it. That is, an optimal merge tree does not contain a node that represents a prefix of the stream if this prefix does not start at t_i for some $1 \leq i \leq n$. We first prove a lemma that shows that adding nodes to a merge tree always increases the merge cost.

LEMMA 2.4. *Let T be a merge tree and let $x \in T$ be one of its nodes. Then there exists a merge tree T' on the nodes $T - \{x\}$ such that* $\mathrm{Mcost}(T') < \mathrm{Mcost}(T)$.

Proof. Assume first that x is a leaf. Then let T' be T without x. We get that $\text{Mcost}(T) \geq \text{Mcost}(T') + \ell_T(x)$ and therefore $\text{Mcost}(T') < \text{Mcost}(T)$. Let x be a nonleaf node of T and let w be its leftmost child in T. The first modification is for node w. If x is not the root of T, then let the parent of x in T be the parent of w in T'. Otherwise, make w the root of T'. The second modification is for the rest of the children of x in T. Make all of them children of w in T' and add them after w's own children, preserving their original order in T. The rest of the nodes maintain in T' their parent-child relationship from T. By (1),

$$\ell_T(v) = \ell_{T'}(v) \qquad \text{for } \; p_T(v) \neq x. \tag{5}$$

That is, the length of any node v that is not a child of x in T remains the same in T' because there is no change in $p(v)$ and $z(v)$. If $v \neq w$ is a child of x, then (1) implies that

$$\ell_T(v) - \ell_{T'}(v) = w - x > 0 \qquad \text{for } \; v \neq w \; \text{ and } \; p_T(v) = x, \tag{6}$$

since w is a later arrival than x. As for w, there are two cases to consider depending on whether w is the root of T' or not. If w is the root of T', then x is the root of T. Hence, $\ell_T(x)$ is not counted in $\text{Mcost}(T)$ and $\ell_{T'}(w)$ is not counted in $\text{Mcost}(T')$. Hence, we have by inequalities (5) and (6)

$$\text{Mcost}(T) - \text{Mcost}(T') = \ell_T(w) + \sum_{(v \neq w) \wedge (p_T(v) = x)} (\ell_T(v) - \ell_{T'}(v)) > 0.$$

If w is not the root of T', then we might have $\ell_{T'}(w) > \ell_T(w)$. However, this is more than compensated for by the inequality

$$\ell_T(x) > \ell_{T'}(w). \tag{7}$$

To see inequality (7), note that $z_T(x) = z_{T'}(w)$ and that $p_T(x) = p_{T'}(w)$; hence by (1), $\ell_T(x) - \ell_{T'}(w) = w - x > 0$. By combining inequalities (5), (6), and (7) we obtain the following:

$$\text{Mcost}(T) - \text{Mcost}(T') = \ell_T(x) + \ell_T(w) - \ell_{T'}(w) + \sum_{(v \neq w) \wedge (p_T(v) = x)} (\ell_T(v) - \ell_{T'}(v)) > 0,$$

which is our desired result. ∎

LEMMA 2.5. *For arrivals $t_1, t_2, \ldots, t_n$ every node (stream) x in an optimal merge tree starts at time t_i for some $1 \leq i \leq n$.*

Proof. Assume to the contrary that there exists a stream x that starts at time t that is not one of the n arrival times $t_1, \ldots, t_n$. By definition, no client needs stream x. Hence, by Lemma 2.4, we could omit node x from T to get a tree T' without x, whose merge cost is smaller than the merge cost of T and is a contradiction to the optimality of T. ∎

Remark. Optimal merge trees also give a lower bound on the bandwidth for the more dynamic event-driven algorithms [18, 19]. A client's receiving procedure, which streams it listens to and when, is determined, in part, by future arrivals. Nonetheless, in the end, the final receiving pattern of a client forms a path in a merge tree. At any point in time only a set of subtrees of the final merge tree is known. Each root of a subtree represents an active stream at that time. When a merge event occurs, the root of some subtree becomes the child of some root in another subtree.

2.2. The full cost. Let F be composed of s merge trees $T_1, \ldots, T_s$. The *full cost* of F is defined as

$$\mathrm{Fcost}(F) = s \cdot L + \sum_{1 \le i \le s} \mathrm{Mcost}(T_i).$$

The above definition is a bit problematic since in the way we define the merge cost it could be the case that a length of a stream is L or larger. Consider the following example. Suppose that the root arrives at time 0 and there are two additional arrivals at times $L-2$ and $L-1$. In one optimal merge tree the third arrival first merges with the second arrival and then both merge with the root; that is, $p(L-1) = L-2$ and $p(L-2) = 0$. The cost of this tree is L for the root, L for the second arrival, and 1 for the third arrival for a total cost of $2L+1$. It is clear that this single merge tree can be considered as a merge forest of two merge trees, the first with one arrival, 0, and the second with two, $L-2$ and $L-1$. A more serious problem is exposed by the following example where the arrival times are 0, $L-3$, and $L-1$. In this case the definition of a merge tree would allow $p(L-1) = L-3$ and $p(L-3) = 0$. In this case the full cost of the merge tree is $2L+3$. Length L for the root 0, length $L+1$ for $L-3$, and length 2 for $L-1$. We have $\ell(L-3)$ greater than L. However, this merge tree is not an optimal merge forest for these three arrivals. The optimal merge forest has two trees, one with root 0 and one with root $L-3$ for a full cost of $2L+2$.

Naturally, we cannot allow the length of any stream to be greater than L. To remedy this problem we define an *L-tree* to be a merge tree in which the length of each stream has length less than or equal to L and the length of the root is L. The first example above is an L-tree, but the second is not. It should be clear that an L-tree with a nonroot x of length L can be split into two L-trees of the same cost by simply making x a new root. An *L-forest* is a merge forest that is composed of L-trees only. For an arrival sequence $t_1, \ldots, t_n$ and stream length L define the *optimal full cost* for the sequence to be the minimum full cost of any L-forest for the sequence. An *optimal L-forest* is one that has optimal full cost.

Our strategy for searching for the optimal L-forest is to consider all possible merge forests as candidates for the optimal. The following lemma shows that this extended search always yields an L-forest as the optimal.

LEMMA 2.6. *Any merge forest F that minimizes* $\mathrm{Fcost}(F)$ *is an L-forest.*

Proof. Define the following split operation on trees. Let T be a merge tree on the arrivals $t_1, \ldots, t_n$. Let $x = t_i$ be a node in the tree. Then the x-split of T creates two trees: T' and T''. T' is rooted at t_1 and contains the arrivals $t_1, \ldots, t_{i-1}$ with the same parent-child relation as in T. T'' is rooted at x and contains the arrivals $t_i, \ldots, t_n$. The parent relation in T'' is defined as follows: Let $y = t_j$ for $i < j \le n$ and let $w = p(y)$ be the parent of y in T. If $w > x$, then $w = p(y)$ in T'' as well. Otherwise, $x = p(y)$ in T''.

Let T be a non-L-tree and let $x \in T$ be a node whose length is $\ell(x) > L$. We claim that

$$\mathrm{Fcost}(T') + \mathrm{Fcost}(T'') < \mathrm{Fcost}(T). \tag{8}$$

We prove this claim by showing that the length of each node, other than x, in T' or T'' is no more than its length in T. Since the length of x is greater than L in T and equal to L in T'', we are done. There are two cases to consider: (i) The length of all the nodes that have the same parent in T' or T'' as they had in T remains the same;

(ii) By Lemma 2.1, the length of all nodes y such that $w = p(y)$ in T but $x = p(y)$ in T'' is reduced since $w < x$.

To prove the lemma, let F be a merge forest that minimizes $\mathrm{Fcost}(F)$. If F is not an L-forest, then there is some merge tree T in F which is not an L-tree. Apply the above split procedure to this tree to obtain a new merge forest with less cost than F. Thus, F must be an L-forest. ∎

3. The optimal algorithm. In this section we give efficient algorithms for finding a merge tree that minimizes the merge cost and for finding a merge forest that minimizes the full cost. For the merge cost case we assume that the root has length infinity and that all the arrivals can merge to it. In the full cost case we assume that the length of a full stream is L. We then search for the best assignment of roots among the n arrivals. Although some of the assignments may lead to non-L-trees (trees in which some of the nodes have length greater than L), by Lemma 2.6 we know that an optimal merge forest is an L-forest.

For the merge cost we present an efficient $O(n^2)$ time algorithm improving the known $O(n^3)$ time algorithm (see [2, 19]). The latter algorithm is based on a straightforward dynamic programming implementation. Our algorithm implements the dynamic programming utilizing the monotonicity property of the recursive definition for the merge cost. For the full cost we use the optimal solution of the merge cost as a subroutine. We describe an $O(nm)$ time algorithm where m is the average number of arrivals in an interval of length $L - 1$ that begins with an arrival.

3.1. Optimal merge cost. Let $t_1, t_2, \ldots, t_n$ be a sequence of arrivals. Define $M(i,j)$ to be the optimal merge cost for the input sequence $t_i, \ldots, t_j$. In a dynamic programming fashion we show how to compute $M(i,j)$. The optimal cost for the entire sequence is $M(1,n)$. By Lemma 2.3 we can recursively define

$$M(i,j) = \min_{i<k\leq j} \{M(i,k-1) + M(k,j) + (2t_j - t_k - t_i)\} \tag{9}$$

with the initialization $M(i,i) = 0$. Using the notation of Lemma 2.3, t_i is the root r, t_j is the last arrival z, and we are looking for the optimal last arrival t_k, which is x, that merges to the root. This recursive formulation naturally leads to an $O(n^3)$ time algorithm using dynamic programming. The following theorem shows that this can be significantly improved.

THEOREM 3.1. *An optimal merge tree can be computed in time $O(n^2)$.*

Proof. To reduce the time to compute the optimal merge cost to $O(n^2)$ we employ monotonicity, a classic technique pioneered by Knuth [33, 34]. Define $r(i,i) = i$ and for $i < j$

$$r(i,j) = \max\{k : M(i,j) = M(i,k-1) + M(k,j) + 2t_j - t_k - t_i\}.$$

That is, $r(i,j)$ is the last arrival that can merge to the root in some optimal merge tree for $t_i, \ldots, t_j$. *Monotonicity* is the property that for $1 \leq i < n$ and $1 < j \leq n$

$$r(i,j-1) \leq r(i,j) \leq r(i+1,j). \tag{10}$$

We should note that there is nothing special about using the max in the definition of $r(i,j)$; the min would yield the same inequality (10). Once monotonicity is demonstrated then the search for the k in (9) can be reduced to $r(i+1,j) - r(i,j-1) + 1$ possibilities from $j - i$ possibilities. Hence, the sum of the lengths of all the search intervals is reduced to $\sum_{1\leq i<n}\sum_{i<j\leq n}(r(i+1,j)-r(i,j-1)+1)=O(n^2)$ from $\sum_{1\leq i<j\leq n}(j-i)=O(n^3)$. This yields an $O(n^2)$ algorithm.

Fortunately, for our problem we can apply the very elegant method of quadrangle inequalities, pioneered by Yao [45] and extended by Borchers and Gupta [6], that leads to a proof of monotonicity. Define $h(i,k,j)=2t_j-t_k-t_i$ which is the third term in (9). Borchers and Gupta show that if h satisfies the following two properties, then monotonicity holds. For $i\leq j<t\leq k\leq l$ and $i<s\leq l$,

1. if $t\leq s$, then $h(i,t,k)-h(j,t,k)+h(j,s,l)-h(i,s,l)\leq 0$ and $h(j,s,l)-h(i,s,l)\leq 0$;
2. if $s\leq t$, then $h(j,t,l)-h(j,t,k)+h(i,s,k)-h(i,s,l)\leq 0$ and $h(i,s,k)-h(i,s,l)\leq 0$.

In our case, both four-term sums are identically zero, while $h(j,s,l)-h(i,s,l)=t_i-t_j\leq 0$ and $h(i,s,k)-h(i,s,l)=2(t_k-t_l)\leq 0$.

As a byproduct of the computation of $r(i,j)$ we can recursively compute the optimal merge tree using the recursive characterization of Lemma 2.3. We define a recursive procedure for computing an optimal merge tree for the input $t_i,\ldots,t_j$ as follows. If $i=j$, then return the tree with one node labeled t_i. Otherwise, recursively compute optimal merge trees T' for the input $t_i,\ldots,t_{r(i,j)-1}$ and T'' for $t_{r(i,j)},\ldots,t_j$, then attach the root of T'' as an additional last child of the root of T' and return the resulting tree. This procedure is then called for the input $t_1,\ldots,t_n$ to get the final result. With an elementary data structure, and with $r(i,j)$ already computed for $1\leq i\leq j\leq n$, the construction of the optimal merge tree can be done in linear time. ▯

We conclude this subsection with an upper bound on the merge cost of an arrival sequence $t_1,t_2,\ldots,t_n$. Denote by $N=t_n-t_i$ the span of the arrivals. We are looking for an upper bound that depends only on N and n and not on the sequence itself. In the following theorem we establish an $O(N\log n)$ upper bound based on a full binary merge tree.

THEOREM 3.2. *The optimal merge cost is* $O(N\log n)$.

Proof. Using the notation of this subsection, we prove that

$$M(i,j)\leq c(t_j-t_i)\log_2(j-i+1)$$

by induction on $h=j-i$, for some constant $c\geq 4$. For the rest of the proof we omit the base 2 from the log function. The theorem follows by choosing $i=1$ and $j=n$. The claim trivially holds for $h=0$. For $h=1$ the claim holds for $c=1$ since $M(i,i+1)=t_{i+1}-t_i$. Assume $h\geq 2$ and that the claim holds for $1,\ldots,h-1$. We distinguish between the cases of an odd h and an even h. In both cases assume that $j-i=h$ for some $1\leq i<j\leq n$.

An odd h.

$$\begin{aligned}
M(i,j) &\leq M\left(i,(i+j-1)/2\right)+M\left((i+j+1)/2,j\right)+2t_j-t_{(i+j+1)/2}-t_i\\
&\leq c(t_{(i+j-1)/2}-t_i)\log\left((h+1)/2\right)+c(t_j-t_{(i+j+1)/2})\log\left((h+1)/2\right)+2(t_j-t_i)\\
&\leq c(t_j-t_i)\log\left((h+1)/2\right)+2(t_j-t_i)\\
&\leq c(t_j-t_i)\log(h+1)-(c-2)(t_j-t_i)\\
&\leq c(t_j-t_i)\log(h+1).
\end{aligned}$$

The first inequality is based on (9). The second inequality is by the induction hypothesis and by the fact that $t_{(i+j+1)/2}>t_i$. The third inequality is valid since $t_j-t_i\geq(t_{(i+j-1)/2})-(t_i+t_j-t_{(i+j+1)/2})$. The fourth inequality is implied since $\log((h+1)/2)=\log(h+1)-1$. Finally, the last inequality holds for $c\geq 2$.

An even h.

$$\begin{aligned}
M(i,j) &\le M\left(i,(i+j-2)/2\right) + M\left((i+j)/2, j\right) + 2t_j - t_{(i+j)/2} - t_i \\
&\le c(t_{(i+j-2)/2} - t_i)\log\left(h/2\right) + c(t_j - t_{(i+j)/2})\log\left((h+2)/2\right) + 2(t_j - t_i) \\
&\le c(t_j - t_i)\log\left((h+2)/2\right) + 2(t_j - t_i) \\
&\le c(t_j - t_i)\log(h+2) - (c-2)(t_j - t_i) \\
&\le c(t_j - t_i)\log(h+1) + 0.5c(t_j - t_i) - (c-2)(t - j - t_i) \\
&\le c(t_j - t_i)\log(h+1) - (0.5c - 2)(t - j - t_i) \\
&\le c(t_j - t_i)\log(h+1).
\end{aligned}$$

The first inequality is based on (9). The second inequality is by the induction hypothesis and by the fact that $t_{(i+j)/2} > t_i$. The third inequality is valid since $t_j - t_i \ge (t_{(i+j-1)/2} - t_i) + (t_j - t_{(i+j+1)/2})$ and $\log((h+2)/2) \ge \log(h/2)$. The fourth inequality is implied since $\log((h+2)/2) = \log(h+2) - 1$. The fifth inequality is due to the fact that $\log_2(h+2) \le \log_2(h+1) + 0.5$ for $h \ge 2$. Rearranging terms implies the sixth inequality. Finally, the last inequality holds for $c \ge 4$. ∎

3.2. Optimal full cost. The optimal algorithm for full cost uses the optimal algorithm for merge cost as a subroutine. Let $t_1, t_2, \ldots, t_n$ be a sequence of arrivals and let L be the length of a full stream. We know that a full stream must begin at t_1; then there are two possible cases in an optimal solution. Either all the remaining streams merge to this first stream or there is a next full stream t_k for some $k \le n$. In the former case, the optimal full cost is simply $L + M(1,n)$. In the latter case, the optimal full cost is $L + M(1, k-1)$ plus the optimal full cost of the remaining arrivals $t_k, \ldots, t_n$. In both cases, the last arrival to merge to the first stream must be within $L-1$ of the first stream. That is, in the former case $t_n - t_1 \le L - 1$ and in the latter case $t_{k-1} - t_1 \le L - 1$.

For $1 \le i \le n$, define $G(i)$ to be the optimal full cost for the last $n - i + 1$ arrivals $t_i, \ldots, t_n$. By the analysis above, we can define $G(n+1) = 0$ and for $1 \le i \le n$

$$G(i) = L + \min\left\{M(i, k-1) + G(k) : i < k \le n+1 \text{ and } t_{k-1} - t_i \le L - 1\right\}. \tag{11}$$

The order of computation is $G(n+1), G(n), \ldots, G(1)$. The optimal full cost is $G(1)$. This analysis leads us to the following theorem.

THEOREM 3.3. *An optimal L-forest can be computed in time $O(nm)$ where m is the average number of arrivals in an interval of length $L-1$ that begins with an arrival.*

Proof. We begin by giving an algorithm for computing the optimal full cost and then show how it yields an algorithm to construct an optimal merge forest. By Lemma 2.6 this optimal merge forest is an L-forest. The optimal full cost algorithm proceeds in two phases. In the first phase we compute the optimal merge cost $M(i,j)$ for all i and j such that $0 \le t_j - t_i \le L - 1$, so that these values can be used to compute $G(i)$. In the second phase we compute $G(i)$ from $i = n$ down to 1 using (11). Define m_i to be the cardinality of the set $\{j : 0 \le t_j - t_i \le L - 1\}$ and define m to be the average of the m_i's, that is, $m = \sum_{i=1}^{n} m_i / n$. The quantity m can be thought of as the average number of arrivals in an interval of length $L-1$ that begins with an arrival.

We argue that each of the two phases can be computed in $O(nm) = O\left(\sum_{i=1}^{n} m_i\right)$ time. This is mostly a data structure issue because the number of additions and subtractions in the two phases is bounded by a constant times $\sum_{i=1}^{n} m_i$. To facilitate the computations we define an array $A[1..n]$ of arrays. The array $A[i]$ is indexed from 0 to $m_i - 1$. Ultimately, the array entry $A[i][d]$ will contain $M(i, i+d)$ for $1 \leq i \leq n$ and $0 \leq d < m_i$. Initially, $A[i][0] = 0$ and $A[i][d] = \infty$ for $1 \leq d < m_i$. In phase one, a dynamic program based on (9) can be used to compute the ultimate value of $A[i][d] = M(i, i+d)$. Here, a specific order is required for the computation of all the nm entries in the array $A[1..n]$. Using the monotonicity property, it can be done in time $O(nm)$. In phase two, we use the array A to access the value $M(i, k-1)$ when it is needed. Since the minimization in (11) ranges over at most m_i values, the time of phase two is bounded by a constant times $\sum_{i=1}^{n} m_i$. Hence both phases together run in time $O(nm)$.

We have already seen how to construct an optimal merge tree, so all that is left is to identify the full streams. This is done inductively using the values $G(i)$ for $1 \leq i \leq n$ that we have already computed. We know that t_1 is a full stream. Suppose that we know the first $j \geq 1$ full streams that are indexed $f_1, f_2, \ldots, f_j$. We want to determine if f_j is the last full stream, or that the next full stream is indexed f_{j+1}. Find the smallest k such that

$$G(f_j) = L + M(f_j, k-1) + G(k),$$

where $f_j < k \leq n+1$ and $t_{k-1} - t_{f_j} \leq L - 1$. If $k = n+1$, then f_j is the last full stream. If $k < n+1$, then the next full stream is indexed $f_{j+1} = k$. When we are done, suppose there are s full streams which start at the arrivals indexed $f_1, f_2, \ldots, f_s$. We then compute s merge trees where the ith merge tree is for inputs $t_{f_i}, \ldots, t_{f_{i+1}-1}$ if $i < s$ and for inputs $t_{f_s}, \ldots, t_n$ if $i = s$. Given that $G(i)$ for $1 \leq i \leq n+1$ and $M(i,j)$ and $r(i,j)$ for $t_j - t_i \leq L - 1$ are already computed, then the time to compute the sequence $f_1, f_2, \ldots, f_s$ and to compute the merge trees rooted at these arrivals is $O(\sum_{i=s}^{n} m_{f_i})$ which is $O(n)$. $\square$

We now compute an upper bound on the full cost of an arrival sequence $t_1, t_2, \ldots, t_n$, where $n \geq 2$. This time we are looking for an upper bound that depends only on $N = t_n - t_1$, n, and L and not on the sequence itself. Define $\rho = n/N$ to be the density of the n arrivals. We have $0 < \rho \leq 1$. If ρ is near zero, then there are very few arrivals over the span N, so we would expect the optimal full cost to be $O(nL)$. On the other hand, if ρ is large, we would expect a lot of merging to occur, reducing the full cost considerably. This intuition is quantified in the following theorem.

THEOREM 3.4. *The optimal full cost is $O(nL)$ for any values of n and N. The optimal full cost is $O(N \log(\rho L))$ for $\rho \geq \alpha/L$ for some positive constant α.*

Proof. The first statement of the theorem is true for any solution since in the worst case each arrival gets a full stream for a total cost of nL. This is optimal if any two arrivals are more than L apart.

To prove the second statement of the theorem, assume that the optimal full cost is obtained by the L-forest F that contains s L-trees. Let the cardinalities of these trees be $m_1, m_2, \ldots, m_s$, where $m_i \geq 1$ for $1 \leq i \leq s$ and $\sum_{i=1}^{s} m_i = n$. It follows that

$$\text{Fcost}(F) = sL + \sum_{i=1}^{s} \text{Mcost}(m_i).$$

By Theorem 3.2 there is a constant c ($c = 4 \log_2 e$ is sufficient) such that

$$\text{Fcost}(F) \leq sL + cL \sum_{i=1}^{s} \log_e m_i.$$

The convexity of the function $\log_e$ implies that

$$\text{Fcost}(F) \leq sL + cL\sum_{i=1}^{s}\log_e(n/s) = sL + csL\log_e(n/s) = sL(c\log_e(n/s)+1). \tag{12}$$

The expression $sL(c\log_e(n/s)+1)$ as a function of s is concave and, by calculus, achieves a global maximum of $ce^{-1+1/c}nL$ at $s = e^{-1+1/c}n$.

We now show a natural upper bound on s:

$$s \leq \frac{4N}{L}. \tag{13}$$

To see this we argue that there cannot be three full streams in an interval of length $L/2$. In such a case, the last arrival of the second tree is less than $L/2$ slots far from the root of the first tree. One could save cost by merging the root of the second tree to the root of the first tree. The worst case happens when every $L/4+1$ slots there is a new stream.

We conclude the proof of the second statement by letting $\alpha = 4e^{1-1/c}$ and assuming that $\rho \geq \alpha/L$. It follows that $e^{-1+1/c}n \geq 4N/L$. The concavity of the Fcost as a function of s implies that Fcost(F) is bounded above when $s = 4N/L$ which is the maximum value for s by (13). By plugging this value for s into (12) we get

$$\begin{aligned}\text{Fcost}(F) &\leq \frac{4N}{L}L\left(c\log_e\frac{n}{4N/L}+1\right)\\ &= O(N\log(\rho L)). \qquad \square\end{aligned}$$

The statement of Theorem 3.4 seems to be different from the statement of Theorem 3.2. Here the performance depends on L as well. Nevertheless, when ρ is large, the term ρL, which is close to the average number of arrivals in an interval of length L, is analogous to n.

We conclude this section by comparing analytically the performance of the traditional batching with the performance of the optimal full cost using the upper bound of Theorem 3.4.

THEOREM 3.5. *There is a positive constant α such that if $\rho \geq \alpha/L$, then batching with stream merging is $\Omega(\rho L/\log(\rho L))$ better than batching alone.*

Proof. Choose α by Theorem 3.4. The cost of batching the n full streams is nL. Therefore, by Theorem 3.4, the ratio between the performance of batching alone and batching with stream merging is

$$\Omega\left(\frac{nL}{N\log(\rho L)}\right) = \Omega\left(\frac{\rho L}{\log(\rho L)}\right). \qquad \square$$

If $\rho < \alpha/L$, then the gain is at most a constant factor in using batching with stream merging over batching alone.

4. Limited buffer size.

In this section we show how to adapt our solution to the case in which each client has a limited buffer size for storing later parts of streams. Let B be the maximum buffer size. Clients start viewing the stream immediately while being able to receive data from at most two streams. Therefore, if a client has b parts in its buffer it must have viewed the first b parts of the stream. Hence, clients never need a buffer of size more than $\lfloor L/2 \rfloor$. In this section, we assume that $B < \lfloor L/2 \rfloor$ and we modify the algorithms accordingly.

Suppose that arrival x belongs to the merge tree T that is rooted at $r < x$. Assume further that T is an L-tree (the length of all nonroot nodes in the tree is less than or equal to L). Our goal is to calculate $b(x)$, the buffer size required by clients that arrive at time x. These clients base their receiving procedure only on earlier arrivals; therefore in calculating $b(x)$, it is enough to consider the merge tree T without all the arrivals after x. Define $T(x)$ to be this tree; in particular, x is the last arrival in $T(x)$.

LEMMA 4.1. *The buffer size required by clients arriving at x in the merge L-tree T rooted at r is*

$$b(x) = \min\{x - r, L - (x - r)\}.$$

Proof. We distinguish between the following two cases.

Case 1. Assume $0 < x - r \leq \lfloor L/2 \rfloor$. Let y be the ancestor of x in $T(x)$ (could be x itself) that is the child of the root r. It follows that x merges to r at time $y + \ell(y)$ that is the end time of the stream that was initiated at y. Now, $\ell(y) = (x-y)+(x-r)$ by (3) which implies that x merges to r at time $2x-r$. At this time x spent $(2x-r)-x = x-r$ slots receiving data from two streams and from this time on x receives data from only one stream. Hence, $b(x) = x - r$.

Case 2. Assume $\lfloor L/2 \rfloor < x - r \leq L - 1$. By the assumption T is an L-tree and hence the length of all the ancestors of x is strictly less than L. It follows that x receives the Lth part of the stream from the root and stops buffering after time $r + L$ which is the end time of the stream initiated at the root. This implies that x buffers exactly $L + r - x$ parts of the stream.

The lemma follows since if $x - r \leq \lfloor L/2 \rfloor$, then $x - r < L - (x - r)$ and if $\lfloor L/2 \rfloor < x \leq L - 1$, then $L - (x - r) < x - r$. ☐

We are now ready to describe the optimal algorithm for the full cost assuming $B < \lfloor L/2 \rfloor$. Let $t_1, t_2, \ldots, t_n$ be the sequence of arrivals. The only change is in the definition of $G(i)$ in (11). In this equation the search for the minimum value was for $i < k \leq n+1$ such that $t_{k-1} - t_i \leq L - 1$. In what follows we modify this condition.

LEMMA 4.2. *For $1 \leq i \leq i' \leq n$, let $t_{i'}$ be the last arrival that can merge to t_i assuming t_i is a root. Then either* (i) $t_{i'} - t_i \leq B$, *or* (ii) $L - (t_{i'} - t_i) \leq B$ *and there are no arrivals t_j such that $t_i + B < t_j < t_i + L - B$.*

Proof. Assume first that there exists an arrival t_j such that $t_i+B < t_j < t_i+L-B$. By Lemma 4.1 it follows that if t_j belongs to a tree rooted at t_i then its buffer size must be greater than B. This means that t_j must belong to a tree rooted at a later arrival than t_i. Thus, in this case $t_{i'} - t_i \leq B$. Assume now that there are no arrivals t_j such that $t_i + B < t_j < t_i + L - B$. The same lemma implies that arrivals in the range $[t_i + L - B, t_i + L - 1]$ need buffer size less than or equal to B if they merge to a tree rooted at t_i. Hence, if there exists an arrival t_j such that $t_i + L - B \leq t_j \leq t_i + L - 1$, then $L - (t_{i'} - t_i) \leq B$. Otherwise, $t_{i'} - t_i \leq B$ since arrivals after $t_i + L - 1$ cannot merge to a tree rooted at t_i. ☐

Let $t_{i'}$ be the last arrival that can belong to a merge L-tree rooted at t_i as implied by Lemma 4.2. Define $G_B(i)$ to be the optimal full cost for the last $n - i + 1$ arrivals $t_i, \ldots, t_n$. We can define $G_B(n + 1) = 0$ and for $1 \leq i \leq n$

$$G_B(i) = L + \min\{M(i, k-1) + G_B(k) : i < k \leq n+1 \text{ and } t_{k-1} \leq t_{i'}\}. \tag{14}$$

The order of computation is $G_B(n+1), G_B(n), \ldots, G_B(1)$. The optimal full cost is $G(1)$. The following theorem is a modification of Theorem 3.3.

THEOREM 4.3. *An optimal merge forest can be computed in time $O(nm)$ where m is the average number of arrivals in an interval of length B that begins or ends with an arrival.*

Proof. The proof is almost identical to the proof of Theorem 3.3. The only change is the definition of m. The $O(nm)$ is true since by Lemma 4.2 the search for the minimum in computing $G_B(i)$ is conducted in at most in two intervals each of size B. □

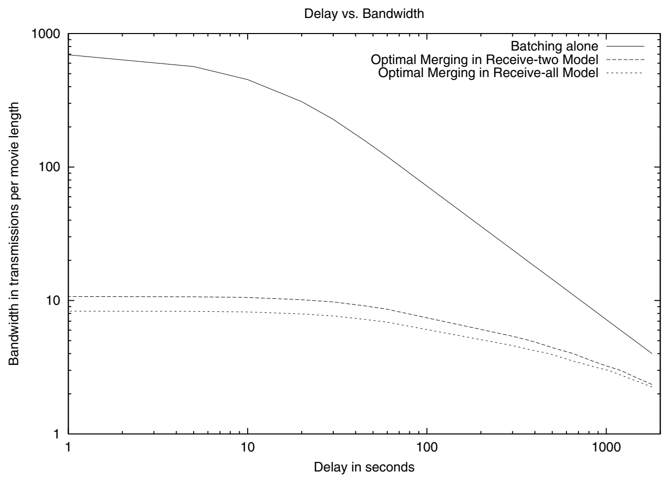

FIG. 6. *Comparison of bandwidth required for batching, batching in the receive-two model, and batching in the receive-all model. The figure plots the bandwidth requirement vs. delay for a* 2*-hour movie, with Poisson arrivals averaging every* 10 *seconds.*

5. The receive-all model.

In this section we consider the receive-all model. In this model a client is capable of receiving data from all the existing streams. Surprisingly, the gain is very little compared to the receive-two model. Experimentally these two models are compared with the traditional batching (receive-one) in Figure 6. We added batching in the receive-all model to the plot of Figure 3. The figure speaks for itself. The rest of the section is devoted to demonstrating analytical comparisons of the full cost. In particular, we show a gain of at most 2. Recall that the gain from the traditional batching to the receive-two model is $\Omega(\rho L/\log(\rho L))$ (for $\rho \geq \alpha/L$ for some α).

We omit some of the details in the proofs of our claims in this section since the proofs are very similar to those in the receive-two model. We first prove some preliminary results as we did in the receive-two model.

In the receive-all model we define merge trees in exactly the same way as the receive-two model. Without going into detail, if $x_0, x_1, \ldots, x_k$ is the path from the root x_0 to node x_k that is the arrival time of a specific client, then the client x_k can receive data from all the streams $x_0, \ldots, x_k$. As in the receive-two model each stream starts at the beginning and runs continuously until it terminates, perhaps early.

Given a merge tree T and a node x, define $\ell_\omega(x)$ to be the minimum length needed to guarantee that all the clients can receive the stream using the receive-all stream merging rules.

LEMMA 5.1. *Let $x \neq r(T)$ be a nonroot node in a tree T. Then*

$$\ell_\omega(x) = z(x) - p(x). \tag{15}$$

In particular, if x is a leaf, then $\ell(x) = x - p(x)$ since $z(x) = x$.

Proof. Let a path from the root r to a leaf y be $r = x_0, x_1, \ldots, x_k = y$. At time x_k the clients arriving at y receive the following parts of the stream: part 1 from x_k, part $1 + (x_k - x_{k-1})$ from x_{k-1}, part $1 + (x_k - x_{k-2})$ from x_{k-2}, and in general part $1 + (x_k - x_i)$ from x_i. In particular they receive part $1 + (x_k - x_0)$ from the root. This means that the stream at x_k must last for at least $x_k - x_{k-1}$ slots in order for the clients arriving at y to receive parts $[1, (x_k - x_{k-1})]$. Since the only stream that can provide parts $[1 + (x_k - x_{k-1}), (x_k - x_{k-2})]$ to these clients is the one at x_{k-1}, this stream must last for at least $x_k - x_{k-2}$ slots. In general, since the stream at x_i provides parts $[1 + (x_k - x_i), (x_k - x_{i-1})]$ to these clients, this stream must last for at least $x_k - x_i$ slots.

Now let x be a node in the tree, let $p(x)$ be its parent, and let $z(x)$ be the node representing the last arrival in the subtree rooted at x. The above arguments imply that the stream at x provides parts $[1 + (z(x) - x), (z(x) - p(x))]$ to the clients arriving at $z(x)$. Since a stream is always a prefix of the full transmission, the length of the stream at x must be at least $z(x) - p(x)$. The proof is completed, since by the definition of $z(x)$, no other clients require later parts from the stream at x. ◻

Define $\mathrm{Mcost}_\omega(T)$ to be the sum of $\ell_\omega(x)$ for all x in T except the root. For a merge forest F consisting of merge trees $T_1, \ldots, T_s$, define $\mathrm{Fcost}_\omega(F) = s \cdot L + \sum_{i=1}^{s} \mathrm{Mcost}_\omega(T_i)$, where L is the length of a full stream. Again we have an elegant recursive formula for the merge cost.

LEMMA 5.2. *Let T be a merge tree with root r and last stream z and let x be the last stream to merge to the root of T. Then we have*

$$\mathrm{Mcost}_\omega(T) = \mathrm{Mcost}_\omega(T') + \mathrm{Mcost}_\omega(T'') + (z - r), \tag{16}$$

where T' is the subtree of all arrivals before the last stream to merge to the root of T and T'' is the subtree rooted at the last stream to merge to the root.

Proof. The length of any node in T' and T'' is the same as its length in T. Since the root of T' is the root of T, it follows that x is the only node in $\mathrm{Mcost}(T)$ whose length is not included in $\mathrm{Mcost}(T')$ or $\mathrm{Mcost}(T'')$. The lemma follows, since by Lemma 5.1 the length of x is $z(x) - p(x) = z - r$. ◻

We are now ready to compute the merge cost and then the full cost. Let $t_1, \ldots, t_n$ be a sequence of arrivals. Define $M_\omega(i,j)$ to be the minimum cost of a merge tree in the receive-all model for the input sequence $t_i, \ldots, t_j$. Similar to the way we computed $M(i,j)$ we can compute $M_\omega(i,j)$ using the recursive formulation based on (16):

$$M_\omega(i,j) = \min_{i<k\leq j} \{M_\omega(i,k-1) + M_\omega(k,j)\} + (t_j - t_i) \tag{17}$$

with the initialization $M_\omega(i,i) = 0$ for $1 \leq i \leq n$. The optimal cost for the entire sequence is $M_\omega(1,n)$. Because t_k does not appear as a parameter in (17), we can use the simpler approach of Yao [45] to show monotonicity. As a result we have an $O(n^2)$ time algorithm for computing an optimal merge tree in the receive-all model.

Equations (9) and (17) allow us to give bounds on the gain in optimal merge cost that can be achieved by moving to the receive-all model.

THEOREM 5.3. *For any arrival sequence $t_1, \ldots, t_n$ and $1 \leq i \leq j \leq n$*

$$M_\omega(i,j) \leq M(i,j) \leq 2M_\omega(i,j).$$

Proof. The proof is by induction on $j-i$. If $j-i=0$, then $M(i,j)=M_\omega(i,j)=0$. If $j-i>0$, then let k and h be such that

$$M(i,j) = M(i,k-1) + M(k,j) + 2t_j - t_k - t_i, \tag{18}$$

$$M_\omega(i,j) = M_\omega(i,h-1) + M_\omega(h,j) + t_j - t_i. \tag{19}$$

The following proves the lower bound claim of the theorem.

$$\begin{aligned} M_\omega(i,j) &\le M_\omega(i,k-1) + M_\omega(k,j) + t_j - t_i \\ &\le M(i,k-1) + M(k,j) + t_j - t_i \\ &\le M(i,k-1) + M(k,j) + 2t_j - t_k - t_i \\ &\le M(i,j). \end{aligned}$$

The first inequality follows since $M_\omega(i,j)$ is a minimization. By the induction hypothesis, we get the second inequality. The third inequality is implied since $t_j \ge t_k$. Finally, (18) yields the last inequality.

The following proves the upper bound claim of the theorem:

$$\begin{aligned} M(i,j) &\le M(i,h-1) + M(h,j) + 2t_j - t_h - t_i \\ &\le 2M_\omega(i,h-1) + 2M_\omega(h,j) + 2t_j - t_h - t_i \\ &\le 2M_\omega(i,h-1) + 2M_\omega(h,j) + 2(t_j - t_i) \\ &\le 2M_\omega(i,j). \end{aligned}$$

The first inequality follows since $M(i,j)$ is a minimization. By the induction hypothesis, we get the second inequality. The third inequality is implied since $t_h \ge t_i$. Finally, (19) yields the last inequality. □ In the same way as we did for the receive-two model, define $G_\omega(i)$ to be the optimal full cost in the receive-all model for the sequence $t_i, \dots t_n$, the last $n-i+1$ arrivals. We have $G(n+1)=0$ and for $1 \le i \le n$

$$G_\omega(i) = L + \min\left\{M_\omega(i,k-1) + G_\omega(k) : i < k \le n+1 \text{ and } t_{k-1} - t_i \le L-1\right\}. \tag{20}$$

The order of computation is $G_\omega(n+1), G_\omega(n), \dots, G_\omega(1)$. The optimal full cost is $G_\omega(1)$. Using exactly the same technique as we did for the receive-two model, we achieve an $O(nm)$ algorithm for computing an optimal merge forest in the receive-all model, where m is the average number of arrivals in a interval of length $L-1$ that begins with an arrival.

We now apply Theorem 5.3 to show the same factor of 2 bound on the optimal full cost.

THEOREM 5.4. *For any arrival sequence $t_1, \dots, t_n$ and $1 \le i \le n+1$,*

$$G_\omega(i) \le G(i) \le 2G_\omega(i).$$

Proof. The proof is by a reverse induction from $n+1$ to 1. For $n+1$ we have $G_\omega(n+1) = G(n+1) = 0$. If $i < n+1$, then we proceed in a way similar to the proof of theorem 5.3, by letting $k > i$ and $h > i$ be such that $t_{k-1} - t_i \le L-1$, $t_{h-1} - t_i \le L-1$, and

$$G(i) = L + M(i,k-1) + G(k), \tag{21}$$

$$G_\omega(i) = L + M_\omega(i,h-1) + G_\omega(h). \tag{22}$$

The following proves the lower bound claim of the theorem:

$$\begin{aligned} G_\omega(i) &\le L + M_\omega(i, k-1) + G_\omega(k) \\ &\le L + M(i, k-1) + G(k) \\ &\le G(i). \end{aligned}$$

The first inequality follows since $G_\omega(i)$ is a minimization. The induction hypothesis and Theorem 5.3 imply the second inequality. The last inequality is by (21).

The following proves the upper bound claim of the theorem:

$$\begin{aligned} G(i) &\le L + M(i, h-1) + G(h) \\ &\le 2L + 2M_\omega(i, h-1) + 2G_\omega(h) \\ &\le 2G_\omega(i). \end{aligned}$$

The first inequality follows since $G(i)$ is a minimization. The induction hypothesis, Theorem 5.3, and the fact that L is positive imply the second inequality. The last inequality is by (22). □

The factor of 2 is not at all tight for optimal full cost. For example, if $L = 2$, then it can be shown that the optimal full cost in the receive-two model is identical to the optimal full cost in the receive-all model.

6. Conclusions.

In this paper, we addressed the problem of designing efficient off-line algorithms to compute the optimal stream merging in media-on-demand systems. In a stream merging system, clients are assigned to receive data from streams that transmit a popular media where they are capable of receiving data from two streams simultaneously. When clients arrive, they get a receiving program that instructs them from which streams to receive data and when. Their program is independent of later arrivals and is simple. Streams are broadcast by the server each time clients request to view the transmission. However, very few of the streams are full streams, thus allowing the savings in the total utilized bandwidth per one popular media. The main advantage of stream merging is its flexibility. With stream merging, it is easier to allocate channels dynamically to various media based on the current demand.

The main objective of this paper was to construct an efficient optimal algorithm. Recall that n is the number of arrivals, L is the length of the full stream, and m is the average number of arrivals in an interval of length $L-1$ that starts with an arrival. With these parameters, we have an $O(nm)$ optimal algorithm. We also showed how to modify our algorithm to be optimal even if the buffer of clients is limited in its size while maintaining the same running time complexity. To obtain these results, we introduced a new abstract model for the stream merging paradigm. Our merge forest model captures all the information regarding the system. We first analyze a single merge tree and then the merge forest itself.

Finally, we considered a stronger model in which clients may receive data from all the existing streams simultaneously. We showed how our techniques extend to this model with less effort. We used these results to show that analytically the gain from the receive-two model to the receive-all model is at most 2, whereas the gain from the traditional batching to the receive-two model is of order $\rho L/\log(\rho L)$, where $\rho \le 1$ is the density of the n arrivals in the time interval between the first and the last arrival and ρ is large enough. This phenomenon was known experimentally and we support it with analytical results.

REFERENCES

[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, *Design and analysis of permutation-based pyramid broadcasting*, Multimedia Systems, 7 (1999), pp. 439–448.

[2] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, *Adaptive piggybacking schemes for video-on-demand systems*, Multimedia Tools Appl., 16 (2002), pp. 231–250.

[3] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, *The maximum factor queue length batching scheme for video-on-demand systems*, IEEE Trans. Computers, 50 (2001), pp. 97–110.

[4] A. Bar-Noy, J. Goshi, R. E. Ladner, and K. Tam, *Comparison of stream merging algorithms for media-on-demand*, Multimedia Systems, 9 (2004), pp. 411–423.

[5] A. Bar-Noy and R. E. Ladner, *Competitive on-line stream merging algorithms for media-on-demand*, J. Algorithms, 48 (2003), pp. 59–90.

[6] A. Borchers and P. Gupta, *Extending the quadrangle inequality to speed-up dynamic programming*, Inform. Process. Lett., 49 (1994), pp. 287–290.

[7] Y. Cai and K. A. Hua, *An efficient bandwidth-sharing technique for true video on demand systems*, in Proceedings of the 7th ACM International Conference on Multimedia, 1999, pp. 211–214.

[8] Y. Cai, K. A. Hua, and K. Vu, *Optimizing patching performance*, in Proceedings of the IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN '99), 1999, pp. 204–215.

[9] S. W. Carter and D. D. E. Long, *Improving video-on-demand server efficiency through stream tapping*, in Proceedings of the 6th International Conference on Computer Communications and Networks (ICCCN '97), 1997, pp. 200–207.

[10] S. W. Carter and D. D. E. Long, *Improving bandwidth efficiency of video-on-demand servers*, Computer Networks, 31 (1999), pp. 111–123.

[11] W. Chan, T. Lam, H. Ting, and W. Wong, *On-line stream merging in a general setting*, Theoret. Comput. Sci., 296 (2003), pp. 27–46.

[12] W. Chan, T. Lam, H. Ting, and W. Wong, *Competitive analysis of on-line stream merging algorithms*, in Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS), 2002, pp. 188–200.

[13] T. Chiueh and C. Lu, *A periodic broadcasting approach to video-on-demand service*, in Proceedings of the SPIE Conference on Multimedia Computing and Networking (MMCN '95), 1995, pp. 162–169.

[14] E. G. Coffman, Jr., P. Jelenković, and P. Momčilović, *The dyadic stream merging algorithm*, J. Algorithms, 43 (2002), pp. 120–137.

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

[16] D. L. Eager, M. Ferris, and M. K. Vernon, *Optimized regional caching for on-demand data delivery*, in Proceedings of the IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN '99), 1999, pp. 301–316.

[17] D. L. Eager and M. K. Vernon, *Dynamic skyscraper broadcasts for video-on-demand*, in Proceedings of the 4th International Workshop on Advances in Multimedia Information Systems (MIS '98), 1998, pp. 18–32.

[18] D. L. Eager, M. K. Vernon, and J. Zahorjan, *Minimizing bandwidth requirements for on-demand data delivery*, IEEE Trans. Knowl. Data Engrg., 13 (2001), pp. 742–757.

[19] D. L. Eager, M. K. Vernon, and J. Zahorjan, *Optimal and efficient merging schedules for video-on-demand servers*, in Proceedings of the 7th ACM International Multimedia Conference, 1999, pp. 199–203.

[20] L. Gao, J. Kurose, and D. Towsley, *Efficient schemes for broadcasting popular videos*, Multimedia Systems, 8 (2002), pp. 284–294.

[21] L. Gao and D. Towsley, *Supplying instantaneous video-on-demand services using controlled multicast*, in Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS '99), 1999.

[22] L. Gao, Z. Zhang, and D. Towsley, *Catching and selective catching: Efficient latency reduction techniques for delivering continuous multimedia streams*, in Proceedings of the 7th ACM International Conference on Multimedia, 1999, pp. 203–206.

[23] L. Golubchik, J. C. S. Liu, and R. R. Muntz, *Reducing I/O demand in video-on-demand storage servers*, in Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '95), 1995, pp. 25–36.

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

[25] K. A. Hua, Y. Cai, and S. Sheu, *Patching: A multicast technique for true video-on-demand services*, in Proceedings of the 6th ACM International Conference on Multimedia, 1998, pp. 191–200.

[26] K. A. Hua, Y. Cai, and S. Sheu, *Exploiting client bandwidth for more efficient video broadcast*, in Proceedings of the 7th International Conference on Computer Communications and Networks (ICCCN '98), 1998, pp. 848–856.

[27] K. A. Hua and S. Sheu, *Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems*, in Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, 1997, pp. 89–100.

[28] L. Juhn and L. Tseng, *Harmonic broadcasting for video-on-demand service*, IEEE Trans. Broadcasting, 43 (1997), pp. 268–271.

[29] L. Juhn and L. Tseng, *Staircase data broadcasting and receiving scheme for hot video service*, IEEE Trans. Consumer Electronics, 43 (1997), pp. 1110–1117.

[30] L. Juhn and L. Tseng, *Fast broadcasting for hot video access*, in Proceedings of the 4th International Workshop on Real-Time Computing Systems and Applications (RTCSA '97), 1997, pp. 237–243.

[31] L. Juhn and L. Tseng, *Enhancing harmonic data broadcasting and receiving scheme for popular video service*, IEEE Trans. Consumer Electronics, 44 (1998), pp. 343–346.

[32] L. Juhn and L. Tseng, *Fast data broadcasting and receiving scheme for popular video service*, IEEE Trans. Broadcasting, 44 (1998), pp. 100–105.

[33] D. E. Knuth, *Optimum binary search trees*, Acta Informa., 1 (1971), pp. 14–25.

[34] D. E. Knuth, *The Art of Computer Programming, Vol.* 3*: Searching and Sorting*, 2nd ed., Addison–Wesley, Reading, MA, 1998.

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

[36] J. Pâris, S. W. Carter, and D. D. E. Long, *A low bandwidth broadcasting protocol for video on demand*, in Proceedings of the 7th International Conference on Computer Communications and Networks (ICCCN '98), 1998, pp. 690–697.

[37] J. Pâris, S. W. Carter, and D. D. E. Long, *A hybrid broadcasting protocol for video on demand*, in Proceedings of the IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN '99), 1999, pp. 317–326.

[38] J. Pâris and D. D. E. Long, *Limiting the receiving bandwidth of broadcasting protocols for video-on-demand*, in Proceedings of the Euromedia Conference, 2000, pp. 107–111.

[39] J. Pâris, D. D. E. Long, and P. E. Mantey, *Zero-delay broadcasting protocols for video on demand*, in Proceedings of the 7th ACM International Conference on Multimedia, 1999, pp. 189–197.

[40] S. Sen, L. Gao, J. Rexford, and D. Towsley, *Optimal patching schemes for efficient multimedia streaming*, in Proceedings of the 9th IEEE International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '99), 1999.

[41] S. Sen, L. Gao, and D. Towsley, *Frame-based Periodic Broadcast and Fundamental Resource Tradeoffs*, Technical report 99–78, University of Massachusetts, Amherst, MA.

[42] Y. Tseng, C. Hsieh, M. Yang, W. Liao, and J. Sheu, *Data broadcasting and seamless channel transition for highly-demanded videos*, in Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '00), 2000, pp. 4E–2.

[43] S. Viswanathan and T. Imielinski, *Pyramid broadcasting for video-on-demand service*, in Proceedings of the SPIE Conference on Multimedia Computing and Networking (MMCN '95), 1995, pp. 66–77.

[44] S. Viswanathan and T. Imielinski, *Metropolitan area video-on-demand service using pyramid broadcasting*, Multimedia Systems, 4 (1996), pp. 197–208.

[45] F. F. Yao, *Efficient dynamic programming using quadrangle inequalities*, in Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC '80), 1980, pp. 429–435.