The Dyadic Stream Merging Algorithm

E.G. Coffman, Jr. Predrag Jelenković Petar Momčilović

Department of Electrical Engineering, Columbia University
New York, New York 10027

E-mail: {egc, predrag, petar}@ee.columbia.edu

We study the stream merging problem for media-on-demand servers. Clients requesting media from the server arrive by a Poisson process, and delivery to the clients starts immediately. Clients are prepared to receive up to two streams at any time, one or both being fed into a buffer cache. We present an on-line algorithm, the ***dyadic stream merging*** algorithm, whose recursive structure allows us to derive a tight asymptotic bound on stream merging performance. In particular, let λ be the Poisson request arrival rate, and let L be the fixed media length. Then the long-time ratio of the expected total stream length under the dyadic algorithm to that under an algorithm with no merging is asymptotically equal to $\frac{3\log(\lambda L)}{2\lambda L}$. Furthermore, we establish the near-optimality of the dyadic algorithm by comparisons with experimental results obtained for an optimal algorithm constructed as a dynamic program. The dyadic algorithm and the best on-line algorithm of those recently proposed differ by less than a percent in their comparison with an off-line optimal algorithm. Finally, the worst-case performance of our algorithm is shown to be no worse than that of earlier algorithms. Thus, the dyadic algorithm appears to be the first near optimal algorithm that admits a rigorous average-case analysis.

Key Words: average-case analysis, stream merging, video-on-demand

1. INTRODUCTION

At a sequence of random times, clients request content streaming from a given media server, e.g., videos from a video-on-demand server, with delivery for each client to begin immediately. To reduce the potentially heavy traffic burden created by these media streams, it is clearly desirable to combine streams of the same content; this can be implemented in practice by using multicast protocols (e.g., see [28]). With a multicast protocol

This work is supported by the NSF Grant No. 0092113.
Journal of Algorithms, 2002, to appear.

in place, a stream sent to a client can be received by all other clients at a minimal possible usage of network resources. To see how this can be done and still preserve immediate-start delivery, we need the following assumptions: clients can receive two streams in parallel and each has a cache for buffering stream content. Although multimedia streaming embraces video, audio, and data streaming, we will stay with video terminology for simplicity.

The basic idea of stream merging can be explained on the following example. Consider a situation in which (i) client C_1 arrives at t_1 and requests a video of duration L and (ii) client C_0 is currently playing the same video from a stream S_0 that began at time $t_0 < t_1$. Client C_1 missed the first $\Delta := t_1 - t_0$ time units of the video from S_0 and that part of the video needs to be sent to C_1 by the server in stream S_1. However, C_1 can make use of stream S_0 by buffering its content for later playback. In that way the stream S_1 can be terminated after Δ time units. This process is called *stream merging*; in the present case, S_1 was discontinued after being "merged" at time $t_1 + \Delta$ with the earlier starting S_0.

Note that the total streaming time has been reduced from $2L$, with no merging, to a minimum achievable value of $L + \Delta$. The total streaming time is a simple and effective measure of bandwidth consumption that we will retain throughout the paper.

Stream merging becomes much more involved as we increase the number of streams that are candidates for merging, because then the number of ways in which merging can be done also increases. For example, consider the case of three clients C_0, C_1, and C_2 arriving at times $t_0 < t_1 < t_2$ and initiating streams S_0, S_1, and S_2 for a video of duration L. Let $\Delta_i = t_i - t_{i-1}$ be the interarrival times. Figure 1 illustrates an example in which the t_i's are given by 0, 3 and 4, and $L = 10$. Consider the ways in which we can merge the streams for all three clients. For the given setup, the two possible merging patterns are shown in Fig. 1. In Fig. 1(a), S_1 and S_2 are merged independently with S_0 as described earlier: C_1 caches S_0 during $[t_1, t_1 + \Delta_1]$ and C_2 caches S_0 during $[t_2, t_2 + \Delta_1 + \Delta_2]$; at the end of the respective intervals S_1 and S_2 are merged with S_0.

The second possibility is first to merge S_2 with S_1 and then S_1 with S_0. This scenario is illustrated in Fig. 1 (b). Figure 2 breaks down Fig. 1 (b) into the individual schedules for C_1 and C_2. Client C_1 plays S_1 and caches S_0 during $[t_1, t_1 + \Delta_1]$. Thereafter, C_1 plays from its buffer which is only fed by S_0 during the last $L - 2\Delta_1$ time units of the video. Client C_2 caches S_1 and plays from S_2 during $[t_2, t_2 + \Delta_2]$, at which point S_2 is discontinued, and play proceeds from C_2's buffer. Client C_2 continues to cache S_1, but in addition, it caches the remainder of S_0 (in a suitably chosen region of the cache where the two buffering operations can not overlap). This continues until $t_2 + \Delta_1 + \Delta_2$ at which point S_1 is shut down and S_0 becomes the only

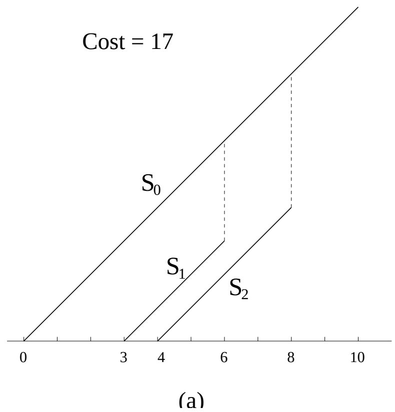

FIG. 1. Stream merging examples. The position of the video runs diagonally. The x-axis represents time. By following the zig-zag lines one obtains which part of the video is being played from which stream. The dashed lines show where the play of the video changes from one stream to another.

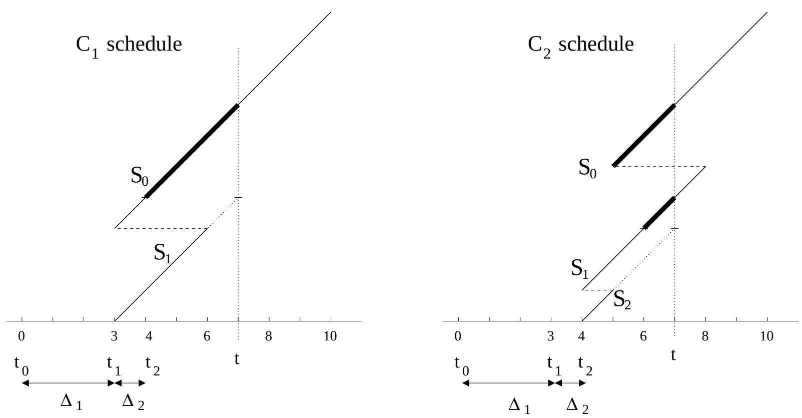

FIG. 2. Individual schedules for the clients C_1 and C_2 in Fig. 1 (b).

active stream while it is supplying the last $L - 2(\Delta_1 + \Delta_2)$ time units of the video to the buffer of C_2. In this process, C_2 has played the first Δ_2 time units of the video directly from S_2, the next $\Delta_1 + \Delta_2$ time units from a cached segment of S_1 and the last $L - \Delta_1 - 2\Delta_2$ time units from a cached segment of S_0.

At any given time, a vertical line in Fig. 2 crosses each of the streams currently being received. Accordingly, in the schedules for C_1 and C_2 the bold lines incident to the vertical lines at time t indicate that the buffer content at time t consists of the corresponding segments of S_0 and S_1.

Note that, although the streaming at C_1 is the same as in the first merging example, S_1 does not terminate at time $t_1 + \Delta_1$ when no longer needed by C_1; the media server must still send S_1 to C_2 until C_2 can switch to S_0, which occurs at $t_2 + \Delta_1 + \Delta_2$. Stream S_1 is "extended" in order to facilitate the requirements of C_2. Without such an extension C_2 would not be in a position to receive all parts of the movie. Note also that the cost (sum of stream durations) of the second merging pattern is 16 as compared to the cost 17 of the first pattern. In general, the best merge pattern for an arrival at time t depends not only on arrival times before t, but also on the arrival times after t. As it will become clear in the next section, for this example the solution, that our dyadic tree algorithm yields, corresponds to Figure 1 (b).

The technique of stream merging originated with Eager, Vernon, and Zahorjan [11, 12] as a model of the *pyramid broadcasting* scheme introduced by Viswanathan and Imielinski [36,37]. This paradigm provides the multicast basis for sharing streams and is built upon the assumption that clients can receive more bandwidth than they need for play-out. The *skyscraper broadcasting* scheme [15,22,31] is another example of these new techniques. A number of related techniques go under the names of batching [1, 9, 10], patching [6, 16, 21], tapping [7,8], and piggy-backing [2, 18, 19, 29] and the general problem has several parameters and useful performance metrics. Other parameters include delay guarantees, receiving bandwidth, server bandwidth, and buffer size [5, 13–15, 17, 20, 23–27, 30–35]. The maximum number of streams is another metric that is of greater interest in certain circumstances. In this setting, the algorithm of this paper has the properties:

- It is on-line, i.e., the media server does not know arrival times in advance.
- It gives a zero-delay guarantee, i.e., all video requests are satisfied immediately.
- It restricts the number of streams being received by a client at any one time to at most two – the *receive-two* model.
- The buffer size can accommodate up to half of the video.

The last two assumptions are justified in the papers by Bar-Noy and Ladner [3, 4], which supply the primary motivation for the work here. In particular, most of the improvement of merging streams is already present in the receive-two model. The $L/2$ buffer size limit comes about because our algorithm does not attempt merging with an existing stream that is already at least half over. As Bar-Noy and Ladner argue, this convenience does not lead to increased average cost even for only moderately large arrival rates. For further discussion of the literature on stream merging, we refer the reader to the mini-survey of [3].

Many excellent numerical/experimental studies have appeared in the stream-merging literature, but the absence of mathematical foundations has stood out, at least until the work in [3, 4], which focuses on competitive, or worst-case, analysis. Here, we give what appears to be the first rigorous average-case analysis of a near-optimal algorithm.

The paper is organized as follows. In Section 2 we present the *dyadic tree* algorithm and state our main results. Section 3 contains numerical experiments that verify the algorithm's performance and conclusions. The proofs of the main results can be found in Section 4.

2. ALGORITHM AND RESULTS

The problem of stream merging can be posed as a problem on trees (see [3,4]). A *merge tree* is a representation of a stream merging diagram, such as those shown in Figure 1. Each stream of the merging diagram corresponds to a node in the associated merge tree. Thus, the number of nodes in the merge tree is equal to the number of requests placed with the server, i.e., the number of clients. If stream S_j is merged directly to an earlier starting stream S_i, then the node associated with S_j is a child of the node associated with S_i. It is convenient to label the nodes with the arrival times of the corresponding streams.

A *root* stream is merged with no other stream, i.e., it is the root in a merge tree. The length of the root stream is always the full length of the video, L. The start rule below provides a simple way to determine which streams are roots, i.e. it defines a sequence of merge trees. Let $t_0, t_1, \ldots$ be the stream starting times.

Start rule: Node t_0 is a root. If t_i is a root, then $t_j = \inf\{t_k : t_k \geq t_i + L/2\}$ is a root. ▮

In other words, the start rule says that a node will be in a given tree only if the root stream of that tree started less the $L/2$ time units ago. As noted earlier, this constraint simplifies the algorithmics; there is a sacrifice in efficiency, but only when traffic is low. For example, suppose we have a root stream starting at time t_0 and an arrival at time t_1 with $t_0 + L/2 < t_1 \leq t_0 + L$. If t_1 is made a descendant of t_0, then no other node arriving after $t_0 + L$ can be merged with t_1 without extending its length to L. Hence, some gain is achieved only if there are no arrivals in the interval $(t_0 + L, 2t_1 - t_0]$. However, this is an unlikely scenario under high traffic load.

When the arrivals are Poisson, the sequence of merge trees becomes a renewal process. This fact allows us to focus our analysis on a single merge tree rooted at t_0. Let $\{t_n\}_{n=0}^{\infty}$ be a sample path of a Poisson process with rate λ on the non-negative reals, and assume for convenience that $t_0 = 0$.

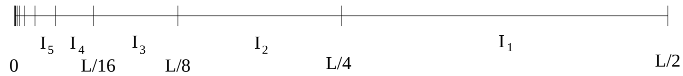

FIG. 3. Dyadic partition of the interval.

The total length of all streams in a merge tree is defined as

$$T \equiv T(L,\lambda) := \sum_{n=0}^{\infty} l_n \mathbf{1}\{t_n < L/2\}, \tag{1}$$

where l_n denotes the length of the stream initiated by the arrival at time t_n; the indicator function $\mathbf{1}\{A\}$ is equal to 1 if A is true and 0 otherwise. By definition $l_0 = L$. The quantity T will measure the effectiveness of stream merging algorithms.

Our new stream merging algorithm is implicit in the following algorithm for constructing merge trees from a given root.

The Dyadic Tree Algorithm: The input is a sequence of $n > 0$ arrival times $t_0, \ldots, t_n$ with $t_0 = 0$, and the output is a tree of n nodes. The arrival at time 0 determines the root. To find the children of the root, first divide the interval $[0, L/2)$ into dyadic subintervals $I_i = [2^{-i-1}L,\ 2^{-i}L)$ with lengths $2^{-i-1}L,\ i = 1, 2, \ldots$, as shown in Figure 3. If I_i contains at least one arrival time, then $t_{(i)}$ denotes the earliest such time; otherwise, $t_{(i)} = 0$. Each $t_{(i)} > 0$ is made a child of the root. Then for each $t_{(i)} > 0$, the algorithm is applied recursively to the interval $[t_{(i)}, 2^{-i}L)$ to determine the subtree rooted at $t_{(i)}$. ▮

It is not difficult to verify that this can be formulated as an on-line algorithm, as we show at the end of this section. In particular, the decision as to where a node t_i should be attached to an existing tree is unaffected by arrivals after time t_i. The following theorem gives our first result, a uniform bound on total stream length. We postpone the proof until Section 4. Throughout the paper we use log to denote $\log_2$.

THEOREM 2.1. *The total cost of the dyadic tree algorithm satisfies*

$$\frac{1}{4}L\log(\lambda L) - \frac{1}{4}L \le \mathbb{E}T(L,\lambda) \le \frac{3}{4}L|\log(\lambda L)| + 3L.$$

Furthermore, it can be shown that the upper bound of the preceding theorem is asymptotically tight for large values of λL. A detailed proof of the next theorem is given in Section 4.

THEOREM 2.2. *The total cost of the dyadic tree algorithm satisfies*

$$\lim_{\lambda L\to\infty} \frac{\mathbb{E}T(L,\lambda)}{L\log(\lambda L)} = \frac{3}{4}.$$

Observe that, by Theorem 2.2, the long-time ratio of the expected total stream length under the dyadic algorithm to that under an algorithm with no merging is asymptotically equal to $3\log(\lambda L)/(2\lambda L)$.

Here we point out that, by Lemma 1 of [3], the length l of the non-root stream initiated at time $t > 0$ is given by

$$l = 2t_l - t - t_p, \tag{2}$$

where t_p is its parent and t_l is the last stream that merges with it. If t is a leaf then $t_l = t$, i.e., $l = t - t_p$.

In order to consider the worst-case performance we examine a slightly different model. This modification is necessary owing to the fact that in the original model the number of requests in $[0, L/2)$ is unbounded, so that the worst case performance is meaningless. Let time be slotted and let the video have a length of $2n$ time slots. We assume that in each of the slots at most one stream can be initiated. According to the start rule a merge tree is being built on n slots. The total stream length achieves its maximum when a stream is initiated in every time slot. In [4] it is proved that the worst-case performance of the optimal algorithm is $\Theta(n\log n)$.

Let $T(2n)$ denote the total stream length for the worst-case merge tree built on n slots, $0, \ldots, n-1$. It is easy to show by induction that $T(n)$ is monotonic in n; hence, one can assume that n is a power of 2. Next, consider two merge trees built on $n/2$ slots each, i.e., $0, \ldots, n/2-1$ and $n/2, \ldots, n-1$. The key fact is that in these two cases only the lengths of streams initiated in the 0th and $n/2$-th slot differ. This follows from the fact that the length of the stream initiated at t depends only on t and starting times of the parent stream and the last stream that merges with it (see (2)). In the first case the lengths of streams initiated at $t = 0$ and $t = n/2$ are $2n$ and $3n/2$, respectively. In the second case the lengths are equal to n. Thus, the difference is $3n/2$ and one obtains $T(2n) = 2T(n) + 3n/2$. The solution to this recurrence has the form $T(n) = \Theta(n\log n)$. Thus, the dyadic algorithm is within a constant factor of optimal in the worst-case. A more detailed numeric comparison of the dyadic algorithm and the optimal algorithm is made in the next section.

We conclude this section with a straightforward on-line implementation of the algorithm.

On-line Dyadic Stream Merging: Let S be a stack with `push` and `pop` operations defined for triples of numbers (t_a, t_r, t_e). Each triple corresponds to a stream: t_a is the time at which the stream was initiated, t_r is the time after which newly arrived streams will not be allowed to merge with it and t_e is the time when the stream terminates.

At time $t = 0$, push the root triple $(0, L/2, L)$ onto S. At time $t < L/2$ of a new request:

1. `pop` the triples (t_a, t_r, t_e) from S until $t_r > t$; at this point let $(\hat{t}_a, \hat{t}_r, \hat{t}_e)$ be the top of the stack,

2. for all but the root triple in S increase the last component by $2t - \hat{t}_e - t_p$, where t_p is the arrival time of the parent of $(\hat{t}_a, \hat{t}_r, \hat{t}_e)$,

3. add the new stream to the stack by performing `push` $(t, t', 2t - \hat{t}_a)$, where $t' = \hat{t}_a + (\hat{t}_r - \hat{t}_a)\max\{2^{-k+1} : 2^{-k}(\hat{t}_r - \hat{t}_a) < t - \hat{t}_a\}$; the stream started at t is the child of the stream started at $\hat{t}_a$.

This procedure uniquely and explicitly defines the merge tree as well as the stream termination times. ∎

3. NUMERICAL RESULTS AND CONCLUSIONS

This section provides a numerical validation of the asymptotic approximation

$$T' \equiv T'(L, \lambda) := L \log(\lambda L).$$

The first example investigates the dependency of the total cost on the length of the stream for fixed values of the arrival rate λ. The parameter values are set within the regions that are plausible for real-life systems. In particular, we set $L = 20i$ min, $i = 1, \ldots, 9$ and plot the ratio $\mathbb{E}T/T'$ in Figure 4, where $\mathbb{E}T$ is obtained by simulating 10,000 trees for each set of values. Points marked with "o", "+" and "x" correspond to λ^{-1} equal to 5, 20, and 60 sec, respectively. Note that for $(\lambda^{-1}, L) = (60\text{s}, 20\text{min})$, the merge tree consists of only 11 nodes on average.

In the second example we fix L and look at $\mathbb{E}T(\lambda, L)$ as a function of the first argument. The simulation results of $\mathbb{E}T/T'$ are plotted in Figure 5. As in the previous case we simulated 10,000 trees for each point. Values of L are set to 120, 60, and 30 min and denoted respectively by the symbols "o", "+", and "x". Using approximation T' with the appropriate multiplicative factor yields excellent engineering estimates for all reasonable values of L and λ.

Finally, we compare the performance of the dyadic tree algorithm to the performance of the optimal off-line algorithm. The cost of the latter can be

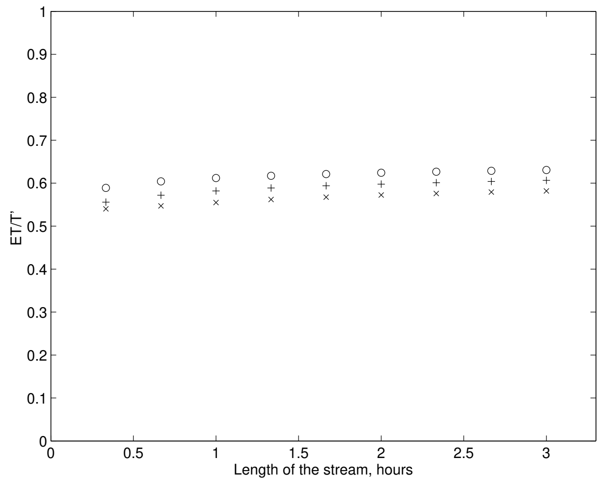

FIG. 4. $\mathbb{E}T/T'$ as a function of the stream length for three values of the arrival rate. Expected interarrival times are 5 sec ("o"), 20 sec ("+") and 60 sec ("x").

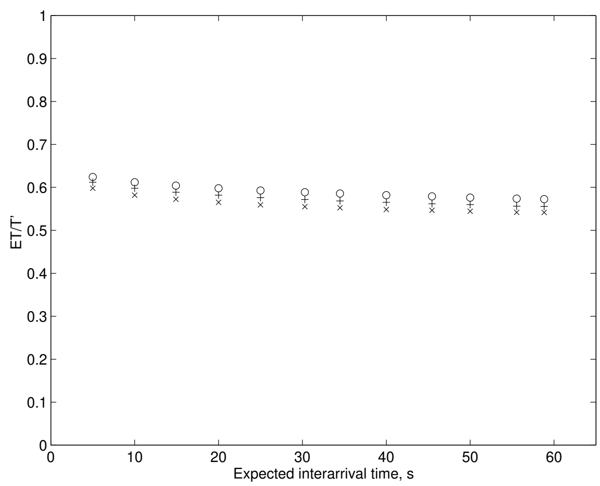

FIG. 5. $\mathbb{E}T/T'$ as a function of the arrival rate for three values of the stream length. The stream length is set to 120 min ("o"), 60 min ("+") and 30 min ("x").

determined by a dynamic program (see [2]). Let $T_{opt}(i,j)$ be the optimal cost of the merge tree for streams initiated at $0 \le t_i < \cdots < t_j < L/2$. The optimal merge tree satisfies the preorder traversal property [4] and, hence,

$$T_{opt}(0,n) = \min_{1\le k\le n} \{T_{opt}(0,k-1) + T_{opt}(k,n) - (L - 2t_n + t_k + t_0)\}$$

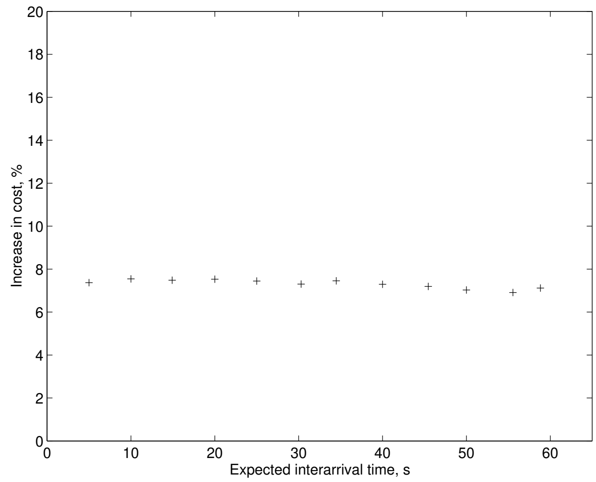

FIG. 6. Performance of the algorithm in comparison with the optimal off-line algorithm. The length of the stream is equal to 2 hours.

with $T_{opt}(i,i) = L$. The last term represents the gain from a merge of optimal trees rooted at t_0 and t_k. We used the fact that the length of the stream t is given by (2).

For numerical comparison, let the length of the video be 2 hours and let the value of the expected interarrival time vary from 5 sec to 60 sec in steps of 5 sec. For every pair (λ, L) we simulated 1,000 trees and based on that computed the average cost for two algorithms. The increase in expected cost when using the dyadic tree algorithm instead of the optimal off-line algorithm is rather small as shown in Figure 6. For all parameter values the increase did not exceed 8%.

In summary, we have been able to prove the tight asymptotic average-case behavior $\frac{3}{4}L\log(\lambda L)$ for the dyadic stream merging algorithm, and to show in addition that its average-case and worst-case performance are comparable to those of the best on-line algorithms known to date.

4. PROOFS

We start by introducing a recursive procedure for labeling the arrival times in $[0, L/2)$. For the purposes of the proof these labels replace the t_i labels. The procedure can be thought of as a function $\mathcal{E}_L : \mathbf{T} \mapsto \omega$ that maps a set $\mathbf{T}$ of arrival times to the space of indices ω. Each index ω consists of a number of digits equal to the depth of the node in the merge tree that corresponds to the given arrival. In general, $\omega = \omega_1\omega_2\ldots\omega_n$, where $\omega_i \in \mathbb{N}$ for $i = 1, 2...$, and the parent of the node labeled ω is a

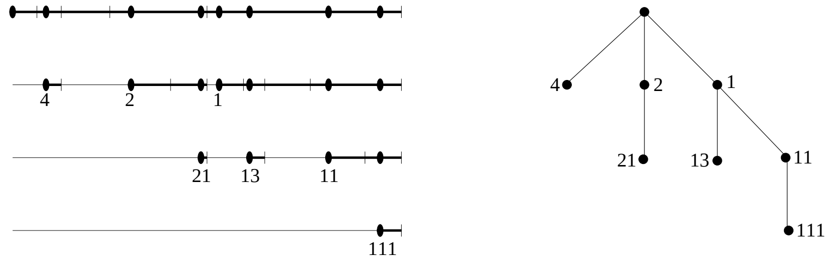

FIG. 7. An illustration of the labeling algorithm. In this example there are seven points that need to be labeled. On the first call of the procedure three points are assigned labels (1,2 and 4). The recursive algorithm is applied until all points are labeled.

node labeled with the prefix $\omega' = \omega_1 \ldots \omega_{n-1}$. The algorithm labels the arrivals as follows. The interval $[0, L/2)$ is divided into dyadic intervals in increasing order from the root as shown in Figure 3. If a point t is the first point in the subinterval I_i then its label is i. Label the rest of the points in $[t, 2^{-i}L)$ recursively by using the parent's label as a prefix for childrens' labels. An example of how the points are labeled is shown in Figure 7.

4.1. Proof of Theorem 2.1

Lower bound: By applying the above labeling procedure, it is not hard to verify that (1) becomes

$$T(L, \lambda) = L + \sum_{n=1}^{\infty} l_n \mathbf{1}\{t_n < L/2\} = L + \sum_{n=1}^{\infty} \sum_{\omega = \omega_1 \ldots \omega_n} l_{\omega_1 \ldots \omega_n}, \qquad (3)$$

where $l_{\omega_1 \ldots \omega_n}$ is the length of the stream starting at the point labeled $\omega_1 \ldots \omega_n$. If for a particular realization of the Poisson process there is no point with label $\omega_1 \ldots \omega_n$, then $l_{\omega_1 \ldots \omega_n} = 0$.

Next, we estimate the expected values of $l_{\omega_1 \ldots \omega_n}$. Let $\eta, \{\eta_n\}_{n=1}^{\infty}$ be a set of i.i.d. exponential random variables with mean λ^{-1}, and consider first the streams that are children of the root, i.e., the streams whose indices consist of a single digit. Given that, for a particular realization of the Poisson process, there exists a stream with label ω_1, its length must be at least $2^{-\omega_1} L/2$ according to (2). Therefore,

$$\begin{aligned} l_{\omega_1} &\geq \frac{L/2}{2^{\omega_1}} \mathbf{1}\left\{\exists n : \frac{L/2}{2^{\omega_1}} \leq t_n < \frac{L/2}{2^{\omega_1+1}}\right\} \\ &\geq \left(\frac{L/2}{2^{\omega_1}} - \inf\left\{t_n - \frac{L/2}{2^{\omega_1}} : t_n > \frac{L/2}{2^{\omega_1}}\right\}\right)^+, \end{aligned}$$

where $(\cdot)^+ \equiv \max(\cdot, 0)$. After taking into account the memoryless property of the Poisson process, we conclude that

$$\mathbb{E}l_{\omega_1} \geq \mathbb{E}\left(\frac{L/2}{2^{\omega_1}} - \eta_1\right)^+.$$

A node with label of form $\omega_1\omega_2$ is a child of the node with label ω_1. Considering the preceding inequality, the recursive nature of the merging algorithm and the size of the problem in which node ω_1 is the root one obtains

$$\mathbb{E}l_{\omega_1\omega_2} \geq \mathbb{E}\left(\frac{\left(\frac{L/2}{2^{\omega_1}} - \eta_1\right)^+}{2^{\omega_2}} - \eta_2\right)^+ = \mathbb{E}\left(\frac{L/2}{2^{\omega_1+\omega_2}} - \eta_2 - \frac{\eta_1}{2^{\omega_2}}\right)^+.$$

The recursive structure of the merging algorithm shows that for a stream with an arbitrary index $\omega_1\omega_2\ldots\omega_n$,

$$\mathbb{E}l_{\omega_1\ldots\omega_n} \geq \mathbb{E}\left(\frac{L/2}{2^{\omega_1+\cdots+\omega_n}} - \eta_n - \sum_{i=1}^{n-1}\frac{\eta_i}{2^{\omega_{i+1}+\cdots+\omega_n}}\right)^+$$

with the understanding that the sum in the above expression is equal to zero if $n = 1$. If $W := \sum_{i=0}^{\infty}\eta_i 2^{-i}$ then for all n and $x \geq 0$

$$\mathbb{P}[W > x] \geq \mathbb{P}\left[\eta_n + \sum_{i=1}^{n-1}\frac{\eta_i}{2^{\omega_{i+1}+\cdots+\omega_n}} > x\right]$$

and, hence, the expected value of an individual stream length is further lower bounded by

$$\mathbb{E}l_{\omega_1\ldots\omega_n} \geq \mathbb{E}\left(\frac{L/2}{2^{\omega_1+\cdots+\omega_n}} - W\right)^+. \tag{4}$$

Now observe that the number of indices with a digit sum equal to k is 2^{k-1}, i.e.,

$$\sum_{n=1}^{\infty}\sum_{\omega=\omega_1\ldots\omega_n}\mathbf{1}\left\{\sum_{i=1}^{n}\omega_i = k\right\} = 2^{k-1}, \tag{5}$$

since the above sum is equal to the number of ways one can partition a set of cardinality k. Rearrange the sum in (3), use the bound (4) and apply (5) to find

$$\begin{aligned}\mathbb{E}T(L,\lambda) &= L + \sum_{k=1}^{\infty} \sum_{\sum \omega_i = k} \mathbb{E}l_{\omega_1 \ldots \omega_n} \\ &\geq L + \sum_{k=1}^{\infty} 2^{k-1} \mathbb{E}\left(\frac{L/2}{2^k} - W\right)^+ \\ &\geq L + \sum_{k=1}^{\infty} 2^{k-1} \left(\frac{L/2}{2^k} - \frac{2}{\lambda}\right)^+,\end{aligned}$$

where the last step follows from Jensen's inequality. Finally, simple manipulations of the preceding bound yield

$$\begin{aligned}\mathbb{E}T(L,\lambda) &\geq L + \frac{L}{4} \sum_{k=1}^{\infty} \left(1 - \frac{2^{k+2}}{\lambda L}\right)^+ \\ &= L + \frac{L}{4} \sum_{k=1}^{\lfloor \log \frac{\lambda L}{4} \rfloor} \left(1 - \frac{2^{k+2}}{\lambda L}\right) \\ &= L + \frac{L}{4} \lfloor \log \frac{\lambda L}{4} \rfloor - \frac{2}{\lambda} \sum_{k=0}^{\lfloor \log \frac{\lambda L}{4} \rfloor - 1} 2^k \\ &\geq \frac{L}{4} \log(\lambda L) - \frac{L}{4},\end{aligned}$$

from which we conclude that the lower bound holds.
Upper bound: Consider the streams that are children of the root. For such streams we have by (2)

$$l_{\omega_1} \leq 3 \frac{L/2}{2^{\omega_1}}. \tag{6}$$

The inequality is tight when there is an arrival right after time $2^{-\omega_1} L/2$ and an arrival just before time $2^{-\omega_1} L$. Next we examine the streams that can be reached from the root in exactly two steps. An upper bound on their length is

$$l_{\omega_1 \omega_2} \leq 3 \left(\frac{\frac{L/2}{2^{\omega_1}} - \inf\{t_n - \frac{L/2}{2^{\omega_1}} : t_n > \frac{L/2}{2^{\omega_1}}\}}{2^{\omega_2}} \right)^+, \tag{7}$$

whereupon the memoryless property of the Poisson process gives

$$\mathbb{E}l_{\omega_1 \omega_2} \leq 3\mathbb{E} \left(\frac{L/2}{2^{\omega_1 + \omega_2}} - \frac{\eta_2}{2^{\omega_2}} \right)^+.$$

Note that (6) and (7) are of the same form. In the first inequality the size of the problem is $L/2$ while in the second the size is $(2^{-\omega_1}L/2 - \inf\{t_n - 2^{-\omega_1}L/2 : t_n > 2^{-\omega_1}L/2\})^+$. Since the merging algorithm is recursive, for streams that have depth $n \geq 2$ in the merge tree one can conclude that

$$\begin{aligned}\mathbb{E}l_{\omega_1\ldots\omega_n} &\leq 3\mathbb{E}\left(\frac{L/2}{2^{\omega_1+\cdots+\omega_n}} - \sum_{i=2}^{n}\frac{\eta_i}{2^{\omega_i+\cdots+\omega_n}}\right)^+ \\ &\leq 3\mathbb{E}\left(\frac{L/2}{2^{\omega_1+\cdots+\omega_n}} - \frac{\eta}{2^{\omega_n}}\right)^+. \qquad (8)\end{aligned}$$

Recall (5) in order to verify that the number of indices with the digit sum k and last digit i is equal to 2^{k-i-1}, i.e., for $1 \leq i \leq k-1$

$$\sum_{n=2}^{\infty}\sum_{\omega=\omega_1\ldots\omega_n}\mathbf{1}\left\{\sum_{j=1}^{n}\omega_j = k,\ \omega_n = i\right\} = 2^{k-i-1}. \qquad (9)$$

The length of the root stream is always L so (3), (6), (8) and (9) yield

$$\begin{aligned}\mathbb{E}T(L,\lambda) &= L + \sum_{\omega_1=1}^{\infty}\mathbb{E}l_{\omega_1} + \sum_{k=2}^{\infty}\sum_{\omega_n=1}^{k-1}\mathbb{E}l_{\omega_1\ldots\omega_n}\mathbf{1}\left\{\sum_{j=1}^{n}\omega_j = k\right\} \\ &\leq L + 3\sum_{k=1}^{\infty}\frac{L/2}{2^k} + 3\sum_{k=2}^{\infty}\sum_{i=1}^{k-1}2^{k-i-1}\mathbb{E}\left(\frac{L/2}{2^k} - \frac{\eta}{2^i}\right)^+ \\ &\leq \frac{5}{2}L + \frac{3}{4}L\sum_{k=2}^{\infty}\sum_{i=1}^{k-1}2^{-i}\mathbb{E}\left(1 - \eta\frac{2^{k+1-i}}{L}\right)^+.\end{aligned}$$

A simple computation shows that $\mathbb{E}(1-\eta)^+ = 1 - \lambda^{-1}(1 - \exp(-\lambda))$; therefore, by changing the order of summation and setting $m = k+1-i$ one obtains

$$\begin{aligned}\mathbb{E}T(L,\lambda) &\leq \frac{5}{2}L + \frac{3}{4}L\sum_{k=2}^{\infty}\sum_{i=1}^{k-1}2^{-i}\left(1 - \frac{2^{k+1-i}}{\lambda L}\left[1 - e^{-\lambda L 2^{-k-1+i}}\right]\right) \\ &= \frac{5}{2}L + \frac{3}{4}L\sum_{i=1}^{\infty}\sum_{m=2}^{\infty}2^{-i}\left(1 - \frac{2^m}{\lambda L}\left[1 - e^{-\lambda L 2^{-m}}\right]\right) \\ &= \frac{5}{2}L + \frac{3}{4}L\sum_{m=2}^{\infty}\left(1 - 2^{m-\log(\lambda L)}\left[1 - e^{-2^{-m+\log(\lambda L)}}\right]\right). \qquad (10)\end{aligned}$$

Finally, straightforward but tedious calculations show that

$$\sum_{j=1}^{\infty}\left(1-2^{j}\left[1-e^{-2^{-j}}\right]\right)\leq 1/2$$

which in conjunction with bound (10) and the monotonicity of the function $1-2^{x}(1-e^{-2^{-x}})$ yields

$$\begin{aligned}\mathbb{E}T(L,\lambda) &\leq \frac{5}{2}L+\frac{3}{4}L\sum_{j=2-\lceil|\log(\lambda L)|\rceil}^{\infty}\left(1-2^{j}\left[1-e^{-2^{-j}}\right]\right)\\ &\leq \frac{5}{2}L+\frac{3}{8}L+\frac{3}{4}L\sum_{j=2-\lceil|\log(\lambda L)|\rceil}^{0}1\\ &\leq \frac{3}{4}L|\log(\lambda L)|+\frac{23}{8}L.\end{aligned}$$

This concludes the proof. ▮

4.2. Proof of Theorem 2.2

The upper bound is a direct consequence of Theorem 2.1. Below we provide the proof of the lower bound. Let $P_\epsilon \equiv P(\lambda,\epsilon) := 1-e^{-\lambda\epsilon}$ denote the probability of having at least one Poisson arrival in an interval of length ϵ. By conditioning on an arrival in both $(2^{-\omega_1-1}L, 2^{-\omega_1-1}L+\epsilon)$ and $(2^{-\omega_1}L-\epsilon, 2^{-\omega_1}L)$ one obtains from (2)

$$\mathbb{E}l_{\omega_1}\geq P_\epsilon^2\left(\frac{3L/2}{2^{\omega_1}}-3\epsilon\right)^{+}.$$

Extending the above reasoning to the streams with two-digit labels yields a lower bound on their expected lengths

$$\mathbb{E}l_{\omega_1\omega_2}\geq P_\epsilon^3\left(\frac{3L/2}{2^{\omega_1+\omega_2}}-\frac{3\epsilon}{2^{\omega_2}}-3\epsilon\right)^{+}.$$

In the above inequality we conditioned on the position of the stream $\omega_1\omega_2$, its parent and the last stream that will merge to it. Due to the recursive structure of the algorithm, for a stream with an arbitrary label $\omega_1\ldots\omega_n$ the lower bound has the following form

$$\begin{aligned}\mathbb{E}l_{\omega_1\ldots\omega_n} &\geq P_\epsilon^{n+1}\left(\frac{3L/2}{2^{\omega_1+\cdots+\omega_n}}-\sum_{i=2}^{n}\frac{3\epsilon}{2^{\omega_2+\cdots+\omega_n}}-3\epsilon\right)^{+}\\ &\geq P_\epsilon^{n+1}\left(\frac{3L/2}{2^{\omega_1+\cdots+\omega_n}}-6\epsilon\right)^{+}.\end{aligned}$$

Next, the preceding inequality, (1) and (5) result in

$$\begin{aligned}\mathbb{E}T &\geq \sum_{k=1}^{\infty} 2^{k-1} P_\epsilon^{k+1} \left(\frac{3L/2}{2^k} - 6\epsilon\right)^+ \\ &\geq \sum_{k=1}^{\lfloor \log(\lambda L)\rfloor} P_\epsilon^{k+1} \left(\frac{3L}{4} - 3\epsilon 2^k\right) \\ &\geq \frac{3}{4} P_\epsilon^{\log(\lambda L)+1} L \lfloor \log(\lambda L)\rfloor - 6\epsilon\lambda L.\end{aligned}$$

Finally, setting $\epsilon = \lambda^{-1}\log\log(\lambda L)$ and using $\log e > 1$ yield

$$\lim_{\lambda L\to\infty} P_\epsilon^{\log(\lambda L)} = \lim_{\lambda L\to\infty}\left(1 - e^{-\log\log(\lambda L)}\right)^{\log(\lambda L)} = 1$$

and, therefore,

$$\frac{T}{L\log(\lambda L)} \geq \frac{3}{4} P_\epsilon^{\log(\lambda L)+1} \frac{\lfloor \log(\lambda L)\rfloor}{\log(\lambda L)} - 6\frac{\log\log(\lambda L)}{\log(\lambda L)} \longrightarrow \frac{3}{4} \qquad \text{as} \quad \lambda L \to \infty.$$

This concludes our proof. ▮

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewer for helpful comments.

REFERENCES

1. C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On optimal batching policies for video-on-demand storage servers. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS '96)*, 1996.
2. C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On optimal piggyback merging policies for video-on-demand systems. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '96)*, pages 200–209, 1996.
3. A. Bar-Noy and R. Ladner. Competitive on-line stream merging algorithms for media-on-demand. Available at http://www.cs.washington.edu/homes/ladner/papers.html. A conference version appeared in Proceedings of SODA'01, 2000.
4. A. Bar-Noy and R. Ladner. Efficient algorithms for optimal stream merging for media-on-demand. Available at http://www.cs.washington.edu/homes/ladner/papers.html, 2000.
5. Y. Cai and K. A. Hua. An efficient bandwidth-sharing technique for true video on demand systems. In *Proceedings of the 7-th ACM International Multimedia Conference, (MULTIMEDIA '99)*, pages 211–214, 1999.

6. 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)*, pages 204–215, 1999.
7. S. W. Carter and D. D. E. Long. Improving video-on-demand server efficiency through stream tapping. In *Proceedings of the 6-th International Conference on Computer Communication and Networks (ICCCN '97)*, pages 200–207, 1997.
8. S. W. Carter and D. D. E. Long. Improving bandwidth efficiency of video-on-demand servers. *Computer Networks*, 31(1-2):99–111, 1999.
9. 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)*, pages 162–169, 1995.
10. A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *ACM Multimedia Systems Journal*, 4(3):112–121, 1996.
11. D. Eager, M. Vernon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. In *Proceedings of the 5-th International Workshop on Advances in Multimedia Information Systems (MIS '99)*, Indian Wells, CA, October 1999.
12. D. Eager, M. Vernon, and J. Zahorjan. Optimal and efficient mergind schedules for video-on-demand servers. In *Proceedings of the 7-th ACM International Multimedia Conference, (MULTIMEDIA '99)*, pages 199–203, Orlando, FL, November 1999.
13. D. L. Eager, M. Ferris, and M. K. Vernon. Optimized regional caching for on-demand data delivery. In *Proceedings of the Conference on Multimedia Computing and Networking (MMCN '99)*, pages 301–316, 1999.
14. D. L. Eager and M. K. Vernon. Dynamic skyscraper broadcasts for video-on-demand. In *Proceedings of the 4-th International Workshop on Advances in Multimedia Information Systems (MIS '98)*, pages 18–32, 1998.
15. L. Gao, J. Kurose, and D. Towsley. Efficient schemes for broadcasting popular videos. In *Proceedings of the 8-th IEEE International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '98)*, 1998.
16. 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.
17. L. Gao, Z. Zhang, and D. Towsley. Catching and selective catching: efficient latency reduction techniques for delivering continuous multimedia streams. In *Proceedings of the 7-th ACM International Multimedia Conference, (MULTIMEDIA '99)*, pages 203–206, 1999.
18. 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 Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '95)*, pages 25–36, 1995.
19. L. Golubchik, J. C. S. Liu, and R. R. Muntz. Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers. *ACM Multimedia Systems Journal*, 4(3):140–155, 1996.
20. K. A. Hua, Y. Cai, and S. Sheu. Exploiting client bandwidth for more efficient video broadcast. In *Proceedings of the 7-th International Conference on Computer Communication and Networks (ICCCN '98)*, pages 848–856, 1998.
21. K. A. Hua, Y. Cai, and S. Sheu. Patching: a multicast technique for true video-on-demand services. In *Proceedings of the 6-th ACM International Conference on Multimedia (MULTIMEDIA '98)*, pages 191–200, 1998.

22. 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*, pages 89–100, 1997.

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

24. L. Juhn and L. Tseng. Harmonic broadcasting for video-on-demand service. *IEEE Transactions on Broadcasting*, 43(3):268–271, 1997.

25. L. Juhn and L. Tseng. Staircase data broadcasting and receiving scheme for hot video service. *IEEE Transactions on Consumer Electronics*, 43(4):1110–1117, 1997.

26. L. Juhn and L. Tseng. Enhancing harmonic data broadcasting and receiving scheme fo popular video service. *IEEE Transactions on Consumer Electronics*, 44(2):343–346, 1998.

27. L. Juhn and L. Tseng. Fast data broadcasting and receiving scheme for popular video service. *IEEE Transactions on Broadcasting*, 44(1):100–105, 1998.

28. J. Kurose and K. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet.* Addison Wesley, 2001.

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

30. J. Pâris, S. W. Carter, and D. D. E. Long. A low bandwidth broadcasting protocol for video on demand. In *Proceedings of the 7-th International Conference on Computer Communication and Networks (ICCCN '98)*, pages 690–697, 1998.

31. 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)*, pages 317–326, 1999.

32. J. Pâris and D. D. E. Long. Limiting the receiving bandwidth of broadcasting protocols for video-on-demand. In *Proceedings of the Euromedia Conference*, pages 107–111, 2000.

33. J. Pâris, D. D. E. Long, and P. E. Mantey. Zero-delay broadcasting protocols for video on demand. In *Proceedings of the 7-th ACM International Multimedia Conference, (MULTIMEDIA '99)*, pages 189–197, 1999.

34. S. Sen, L. Gao, and D. Towsley. Frame-based periodic broadcast and fundamental resource tradeoffs. Technical Report 99-78, University of Massachusetts Amherst.

35. 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 19-th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '00)*, 2000.

36. 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)*, pages 66–77, 1995.

37. S. Viswanathan and T. Imielinski. Metropolitan area for video-on-demand service using pyramid broadcasting. *ACM Multimedia Systems Journal*, 4(3):197–208, 1996.