Video-on-Demand Based on Delayed-Multicast: Algorithmic Support

N. GLINOS[1], D. B. HOANG[2], C. NGUYEN[2] AND A. SYMVONIS[3]

[1]*Department of Mathematics, University of Ioannina, 45110 Ioannina, Greece*
[2]*Department of Computer Systems, University of Technology, Sydney NSW 2007, Australia*
[3]*Department of Mathematics, National Technical University of Athens, Zografou Campus, 15780 Athens, Greece*
Email: nglinos@cc.uoi.gr, dhoang@it.uts.edu.au, chi@it.uts.edu.au, symvonis@math.ntua.gr

In this paper, we examine algorithmic issues related to the delayed multicast technique for video-on-demand delivery. We introduce the minimum total memory (MTM), minimum total traffic (MTT) and the minimum maximum memory per node (MMMN) delayed-multicast allocation problems. We examine these problems on two networks of practical interest, namely, the chandelier and the broom networks. We provide polynomial time algorithms for solving the MTM and the MTT problems on the chandelier network and the MTM problem on the broom network. We also show that a version of the decision-MMMN problem on a general graph is NP-complete. Finally, we present a heuristic method for obtaining a solution for the MTM problem on tree networks.

Received 10 February 2003; revised 17 February 2004

1. INTRODUCTION

Despite recent advances in processor speed, storage capacity and transmission bandwidth, building a cost-effective and scalable continuous multimedia system still remains a challenge. This is best seen in a common application of multimedia systems which is to provide 'video-on-demand' (VoD). In such a system, a user can request a particular video clip, which may be a movie, lecture or news item, and have that clip delivered to the user's multimedia station. In order to build a functional VoD system many technical problems must first be overcome. A common thread running through these problems is the large amount of bandwidth required, both from I/O and the network.

To address the network bandwidth problem, we proposed in [1] the delayed-multicast technique.[1] The motivation stems from the fact that processing power, disk and memory storage are relatively cheap in comparison with network bandwidth. By utilizing existing computational resources such as processing power, memory and disk space at internal nodes in the path from the server to the users, we can increase the level of concurrency in the system, yet at the same time minimize network traffic. To simplify the presentation, we assume constant bandwidth requirements for all transmitting streams.

Our technique is similar to the idea of multicasting in which rather than sending the same information from the source to each receiver, data packets are copied within the network at fan-out points and sent to multiple receivers. This effectively saves bandwidth in the transmission paths from the source to the fan-out points. Where our technique differs from the traditional multicast is in the fact that we introduce a buffer at fan-out points in the network to service requests with different starting times. The scheduling of each transmission is achieved by delaying (hence the term delayed-multicast) the transmission for the required amount of time using the buffer, such that each request receives its data at its scheduled time.

The idea behind the delayed-multicast technique is illustrated in Figure 1 where four requests for the same movie are made to the video server but at different times. Traditionally this would require the server to send four streams, one per request, to service them (Figure 1a). However, if the request times are known in advance, and by employing 'delayed-multicast', the bandwidth requirement from the video server to the intermediate router can be reduced to just one stream since data packets from the leading stream—the stream with the earliest starting time—can be buffered at the intermediate router to be forwarded on at the appropriate time. This is illustrated in Figure 1b.

The delayed-multicast technique requires that the starting times of all requests are known in advance. Thus, the technique finds direct application to data delivery systems that employ some form of data reservation/ordering. However, the technique can also be applied to the implementation of a VoD system that accepts on-line requests. This is achieved by having a hybrid system where the delayed-multicast

[1]In [1] we used the term 'scheduled-multicast' which, we later discovered, had already been used in many other contexts. To avoid confusion, we changed the name to 'delayed-multicast'.

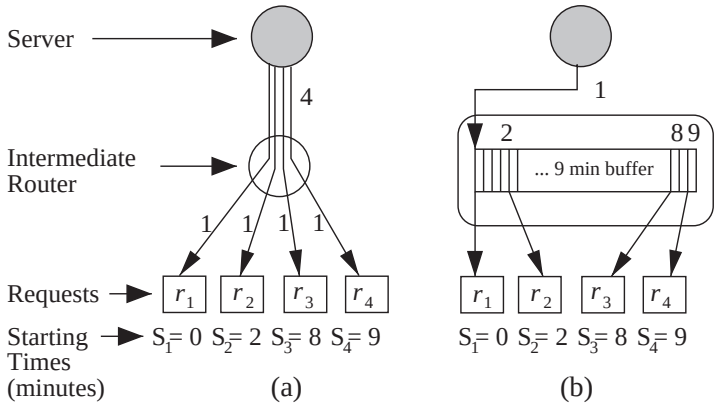

FIGURE 1. (a) An example scenario with four requests of the same video clip going through an intermediate router. (b) An intermediate router employing a buffer to service the four requests with only one transmission stream required coming from the server.

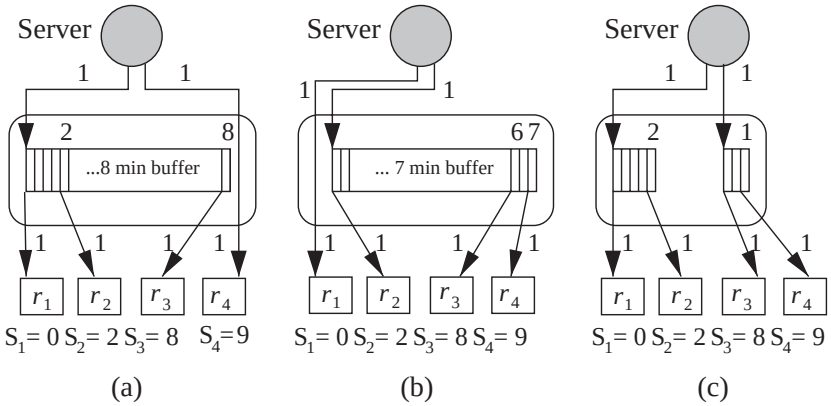

FIGURE 2. Different buffer utilization for the same set of requests.

part of it is responsible for servicing requests that arrived in the past, while a small amount of bandwidth is reserved for servicing requests that arrive on-line. Shortly after the arrival of a new request, the responsibility for servicing the request is passed to the delayed-multicast part of the system which, in the meantime, has rearranged its buffering area accordingly.

The idea of delayed-multicast opens up many interesting research directions spanning the areas of computer algorithms and computer networks. Most of the algorithmic problems are related to tradeoffs between bandwidth and memory. To illustrate this, consider the example of Figure 1. If the bandwidth from the server to the intermediate node is only sufficient for one transmission stream then the only option available is to maintain a buffer of size equal to 9 min of video clip (Figure 1b). However, if the bandwidth available is sufficient for two streams then we have the extra options demonstrated in Figure 2. The illustration shows that by utilizing the extra bandwidth, one can minimize the memory requirement at the router to a total of only 3 min of video clip. An important question is at what point this tradeoff would make the best use of the system resources.

The rest of the paper is organized as follows: Section 2 surveys related work on bandwidth management for on-demand multimedia systems. Section 3 introduces the notation used in the paper and formally defines the examined allocation problems. In Sections 4 and 5, we examine allocation problems on the chandelier and the broom networks respectively. In Section 6, we establish that a version of the minimum maximum memory per node allocation problem is NP-complete even on a two-level broom. In Section 7, we provide a heuristic method for solving an allocation problem on a tree network. We conclude in Section 8 with an open problem raised by our research. A preliminary version of this paper appeared in [2].

2. RELATED WORK

Previous research into bandwidth management for on-demand multimedia delivery can be categorized broadly into two areas: disk I/O and network bandwidth. Both share the common goal of trying to maximize the number of concurrent users in the system given a limited bandwidth; thus many proposed solutions share similar ideas and approaches.

For disk I/O, an important consideration is the poor latency, resulting mainly from slow seek time. To maintain a constant rate of transfer for jitter-free display, it is necessary to keep a buffer in memory for each request. However, as the number

of requests increases, the buffer requirement also increases, resulting in a non-scalable system. By using a 'matrix-based allocation scheme', Özden *et al*. [3] showed how to minimize the disk latency, and therefore minimize the buffer space requirement. Their work was extended in [4] by Garofalakis *et al*. to handle different display rates, retrieval rates and video clip lengths. In addition, they looked at how the matrix-based allocation scheme can be applied to different data layout schemes on disk, namely clustering, where the entire clip is stored on a single disk, and striping, where each clip is de-clustered over all available disks (similar to a RAID file system [5]).

In cases where multiple requests for the same clip differ by a small time interval, system memory can be used to buffer data read from the disk, saving subsequent disk accesses. This technique was introduced by Kamath *et al*. [6] in conjunction with a heuristic for determining when buffer sharing is beneficial. In [7], Shi and Ghandeharizadeh showed that naive use of buffer sharing can in fact degrade system performance by exhausting system memory. Instead they proposed a more stringent heuristic to calculate the threshold at which sharing is beneficial. Their heuristic took into consideration the costs of bandwidth and memory, in addition to memory availability, request inter-arrival times and data rate.

A different but simple approach to overcome the I/O problems is to keep the number of requests to a minimum. This is achieved by broadcasting popular clips on multiple channels to all users on the network who can then individually 'tune' to the channel for their desired video. An increase in the number of users here no longer results in an increase in the number of requests at the server. The obvious disadvantage is that this service does not provide true VoD because a user usually must wait until the scheduled broadcast time in order to watch the movie from the start. In the worst case, the wait-time can be as long as the clip.

In [8], Viswanathan and Imielinski introduced pyramid broadcasting, which aimed to reduce the maximum wait-time to a small fraction of the clip's length. This form of service with small wait-time is often referred to as near VoD (NVoD). With pyramid broadcasting, videos are segmented in K segments of increasing size geometrically and the broadcast bandwidth is evenly divided into K logical channels, with the ith channel being used to broadcast the ith segment of all the videos in a sequential manner. Pyramid broadcasting also requires that the time to download the $(i+1)$th segment must be less than the time to display the ith segment. To display a video, the client downloads and displays the first segment while at the same time buffers the next segment from the second channel to disk. When it is time to display the second segment, the client retrieves it from disk and at the same time tunes-in to the third channel to buffer the third segment on disk. This process continues until all segments have been downloaded and displayed. Since the size of the first segment is small, this minimizes the wait-time before a client can begin watching a video. Aggarwal *et al*. [9], and Hua *et al*. [10] improved on pyramid broadcasting to minimize disk space and I/O bandwidth requirements at the clients' ends. Several schemes related to pyramid broadcasting schemes have been developed. Among them are the fast broadcasting scheme [11], the PAGODA scheme [12] and the recursive frequency splitting scheme [13]. All the above schemes use K channels for the transmission of each video. A way of seamlessly adjusting K for some of the videos that are being broadcast using the fast broadcasting scheme has been proposed by Tseng *et al*. [14].

The disadvantage of pyramid broadcasting and related schemes mentioned above is that they are applicable only in networks where there are dedicated broadcast channels. (In addition, the pyramid-type broadcasting schemes cannot handle the transmission of live events.) They are not suitable for wide-area heterogeneous traffic networks where the bandwidth required for broadcasting is too costly. Instead, a different transmission technique called multicast is used for these networks. A thorough survey of the multicast technique is presented in [15]. Similar to broadcast, multicast requires that all users subscribing to a multicast channel must receive the data at the same time. Thus, no savings can be made if two or more users request the same video clip but with slightly different request times. Batching [16, 17] tries to minimize this problem by grouping requests and serving them together at regular intervals. However, for batching to be effective, it requires a large interval leading to long wait-times for the users. Furthermore, batching can only provide an NVoD type of service.

To provide true VoD, various techniques have been proposed using multicast as the underlying transmission technique. One approach is stream aggregation, which tries to bridge the temporal skew between consecutive multicast streams for the same video clip. This is achieved through rate adaptation [18, 19] where frames are selectively dropped from the transmitting side and, at the receiver's end, interpolation techniques are used to compensate for the loss of frames. Whilst this technique might be feasible for the image quality of the video stream, the sound degradation is much more noticeable and often beyond the tolerance level. A similar idea to stream aggregation is adaptive piggybacking, which was proposed by Golubchik *et al*. [20] and later improved by Aggarwal *et al*. [21]. With adaptive piggybacking, the bridging of the temporal skew is done by speeding-up the display rate of the later stream slightly while at the same time slowing down that for the earlier stream so that they will merge into one stream eventually. This was based on a belief that small deviations of up to 5% from the actual display rate are not perceived by the viewer.

A more recent approach that utilizes the client's disk space to minimize network bandwidth was introduced by Hua *et al*. in [22]. With their method, termed patching, the first set of requests is serviced using a regular multicast. Subsequent requests for the same clip are serviced immediately using a patching multicast. However, rather than subscribing to only one multicast channel, the later clients also subscribe to the nearest earlier regular multicast, using the patching multicast for immediate display and buffering the data from the regular multicast on disk. Once the buffered data is sufficient to bridge the temporal skew between the two multicasts, the

display data is fetched from the buffer and the patching multicast is terminated. An optimal threshold was derived in [23] to determine at what point a new regular multicast should be started for later requests for the same clip. To handle cases where the time difference between the later request and an earlier regular multicast is greater than the client's available buffer, Sen *et al*. introduced a new algorithm called Periodic Buffer Reuse [24].

A requirement of patching is that the client's network bandwidth must be at least twice the bit rate of the clip. This requirement was reduced by Eager *et al*. in [25]. They introduced a modified form of patching, termed bandwidth skimming, that can be employed any time the client's bandwidth is greater than the clip's bit rate—the extra bandwidth is referred to as the skimmed bandwidth. To address the issue of packet loss recovery, Mahanti *et al*. [26] extended the work on bandwidth skimming as well as pyramid broadcasting for broadcast networks, to produce the Reliable Bandwidth Skimming and the Reliable Periodic Broadcast protocols.

An important point to note, which is common to all the above techniques, namely pyramid broadcasting, patching and bandwidth skimming, is that the clients are required to possess large buffer space and the bandwidth in their 'last-mile' connection needs to be greater than (and in most cases at least twice) the clip's playback rate. By pushing the buffering into the network, our technique allows greater bandwidth saving through multi-level network buffering, cheaper set-top boxes and reduces the client's required last-mile bandwidth to be the same as the clip's playback rate.

Similar ideas to the ones presented in this paper have appeared in [27, 28]. Both papers introduce similar video caching techniques based on circular buffers. In [27], Liao and Li introduce the 'split and merge' protocol in order to support the interactive behavior of a true VoD system. In [28], Chan and Tobagi study a distributed server architecture, and examine analytically the trade-off between storage and network channels. Our work studies multilevel caching from a combinatorial perspective, focusing on determining optimal buffer usage for given network channel availability. We are also concerned with the computability of the formulated problems. Finally, the topic of efficient buffer management for Scalable Media-on-Demand has been addressed in [29].

3. NOTATION

We consider a number of video clips that are available for viewing, each of which is identified by a unique clip-id. We assume that all video clips have identical bit-rate for streaming. This assumption allows us the convenience of measuring the capacity of each network link in connections, each connection being capable of carrying exactly one video clip at constant bit-rate. It further allows us to measure the memory used for buffering at the nodes of the network in units of memory-minutes or in frames (assuming that all frames are of identical size in bits).

The network over which the video clips are transmitted is modeled as a weighted graph $G = (V, E)$ where the vertices in V represent the nodes of the network and the edges in E represent the communication links that connect them. With each edge $e \in E$, we associate an integer weight B_e representing the number of connections that can be concurrently established through the edge, i.e. the bandwidth of the communication link measured in units of connections. With each vertex $v \in V$, we associate an integer weight M_v representing the amount of memory (measured in minutes or in frames) available for buffering at vertex v. In the special case where $M_v = \infty$, there are no bounds on the amount of memory we can use for buffering at node v. We assume that the transmission of a frame from one node to its neighbor over a communication link that connects them occurs instantly. The only way to introduce a delay to the frames of a stream is by buffering its contents at the nodes it is passing through.

One node of the network is designated as the server S from which the video streams originate. All other nodes of the network can issue a number of requests. The set of all requests is denoted by R. Each request $r \in R$ is issued by a particular node, denoted by $i(r)$, and concerns the video clip with a particular clip-id, denoted by clip(r). The request also specifies a starting time, denoted by $s(r)$. At time $s(r)$ the first frame of the requested video clip(r) must be delivered by a video stream to node $i(r)$. Subsequently, the video stream will continue providing the frames of the video clip (in the correct order) at a constant rate. Note that after a connection has been established, it is up to the node which issued the request to terminate it. In other words, the length of a video clip is considered to be infinite. Without loss of generality, we assume that the starting time of the earliest request is zero.

REMARK 3.1. *The case where the video is of limited length can be treated by either retransmitting the video-clip, or by caching it at the network nodes. For the latter case, the algorithms presented in the paper can be adapted easily so that the amount of memory used for buffering is proportional to the size of the video clip and does not depend on the starting time of the latest request.*

As mentioned above, over each edge $e \in E$ of the network, we can establish up to B_e connections. For each connection c over an edge $e = (u, v)$, we denote by edge(c) the edge over which the connection is established, i.e. edge(c) $= e$. We also associate with c an integer connection-id, denoted by id(c), such that $1 \leq \text{id}(c) \leq B_e$. Thus, each connection c can be uniquely identified by the tuple (edge(c), id(c)). By orig(c) and dest(c) we denote the nodes at the endpoints of edge e, i.e. orig(c) $= u$ and dest(c) $= v$. By clip(c) we denote the clip-id of the video clip which c is transmitting and by start(c) the time the first frame of the associated video clip(c) was transmitted from node orig(c) to node dest(c). For each node $v \in V$, we denote by Into(v) the set of all connections into node v, i.e. $c \in \text{Into}(v) \Leftrightarrow \text{dest}(c) = v$.

With each connection c, we associate a (shift) buffer that is located at node dest(c). The buffer is denoted by buf(c) and is of size $|\text{buf}(c)|$, measured in minutes of video data. The clip transmitted through a particular connection c is delay-multicasted at node dest(c) with the use of buf(c), which, after introducing the necessary delay, becomes the source of

data for several connections out of node dest(c). For the sake of uniformity, we allow buffers of size zero (0) to cover the case where the clip transmitted through connection c is not delayed at dest(c) and is the source of only one connection out of dest(c). Thus, each connection c (with the exception of the connections out of the server and the connections into the nodes issuing requests) gets its data out of a buffer at node orig(c) and provides data to a buffer at node dest(c). The buffer which provides data to connection c is called the source buffer of c and is denoted by sb(c). The connection which provides data to the buffer sb(c) is called the parent connection of c and is denoted by $p(c)$. Connections out of the server do not have a parent connection.

A set of requests R on a network modeled by a weighted graph G with a designated server node S forms the delayed-multicast resource allocation problem which is denoted by the triplet (R, G, S) (for brevity, we refer to it simply as the allocation problem). Consider the allocation problem $A = (R, G, S)$ and let $r \in R$ be a request. Request r is said to be satisfied if and only if the first frame of the video clip(r) is delivered at node $i(r)$ (which issued the request) precisely at time $s(r)$. More formally, we say that a request r is satisfied if and only if there exists a sequence of connections $C(r) = \{c_1, \ldots, c_k\}$, referred to as the stream of r, which satisfies:

(a) $\text{orig}(c_1) = S$. (The stream starts at the server node S.)
(b) $\text{dest}(c_k) = i(r)$. (The stream arrives at the node which issued request r.)
(c) $\text{start}(c_k) = s(r)$. (The stream arrives on time.)
(d) $p(c_i) = c_{i-1},\ i = 2, \ldots, k$. (Each connection is feeding data to the next.)
(e) $|\text{buf}(c_i)| \leq M_{\text{dest}(c_i)},\ i = 1, \ldots, k$. (There is enough memory.)
(f) $\text{clip}(c_1) = \text{clip}(r)$. (The stream delivers the correct video clip.)

A set of requests R is said to be satisfied if and only if each request $r \in R$ is satisfied. The solution to the allocation problem $A = (R, G, S)$ is a set of connections (together with their associated buffers) $C(A)$ that satisfies:

(a) Each request $r \in R$ is satisfied by a stream $C(r)$ such that $C(r) \subseteq C(A)$.
(b) For each node $v \in V$ it holds that $\sum_{c \in \text{Into}(v)} |\text{buf}(c)| \leq M_v$. (There is enough memory at each node to accommodate the buffers for all incoming connections.)
(c) For each edge $e \in E$ it holds that $|\{c\text{: edge}(c) = e\}| \leq B_e$. (There is enough bandwidth at each edge to accommodate all connections over it.)

(1) *Allocation feasibility problem*. Does there exist a solution for allocation problem A?
(2) *Minimum total memory allocation problem* (*MTM*). Out of all feasible solutions for the allocation problem A identify one solution $C(A)$ such that $\sum_{c \in C(A)} |\text{buf}(c)|$ is minimized.
(3) *Minimum maximum memory per node allocation problem* (*MMMN*). Out of all feasible solutions for the allocation problem A identify one solution $C(A)$ such that $\max_{v \in V} \left\{ \sum_{c \in \text{Into}(v)} |\text{buf}(c)| \right\}$ is minimized.
(4) *Minimum total traffic allocation problem* (*MTT*). Out of all feasible solutions for the allocation problem A identify one solution $C(A)$ such that $|C(A)|$ is minimized.

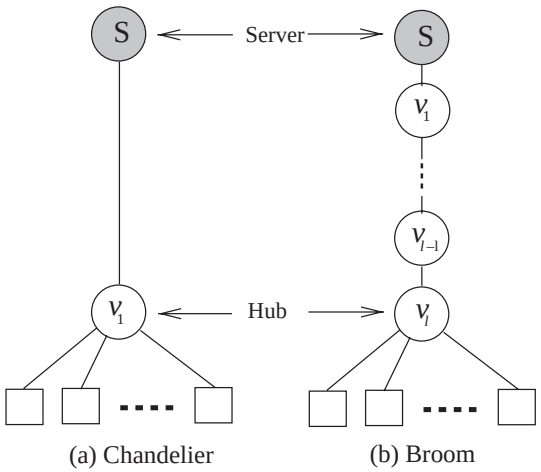

FIGURE 3. The chandelier and the broom tree networks.

Let p be the number of different video clips and $R = R_1 \cup R_2 \cup \cdots \cup R_p$ be the set of requests where R_i consists of all requests for video-clip i, i.e. $R_i = \{r_{i,1}, r_{i,2}, \ldots, r_{i,|R_i|}\}$ where $\text{clip}(r_{i,j}) = i$ for $1 \leq j \leq |R_i|$. For convenience, we assume that the requests in each R_i are ordered with respect to their starting times, i.e. $s(r_{i,j}) \leq s(r_{i,j+1})$ for $1 \leq j < |R_i|$. Denote by $\text{gap}(r_{i,j}) = s(r_{i,j}) - s(r_{i,j-1}),\ 1 < j \leq |R_i|$, the time-gap between request $r_{i,j}$ and the one which precedes it in the sorted order (if it exists).

The allocation problem was defined on a general graph G. We will study the problems on two special tree networks of practical interest, namely the chandelier and the broom. The chandelier network G_{Ch} is illustrated in Figure 3a. It is a three-level tree where the root is the server, the leaves issue the requests, and, memory for buffering purposes can be only placed at the node above the leaves. The node above the leaves is referred to as the hub. The broom network G_{Br} is illustrated in Figure 3b. It is also a tree network with a single hub-node and it can be considered as an extension of the chandelier in the sense that there is a single l-edge simple path, referred as the broom-stick, from the server to the hub. In the case of the chandelier, $l = 1$.

4. RESOURCE ALLOCATION ON THE CHANDELIER TOPOLOGY

Consider the chandelier topology and let B be the bandwidth of edge (S, v_1), i.e. up to B connections can be established from the server to the hub. Without loss of generality, assume that each leaf issues exactly one request, that the start times of the requests for the same video clip are distinct, and that the bandwidth into a leaf is equal to 1 (a leaf issuing multiple

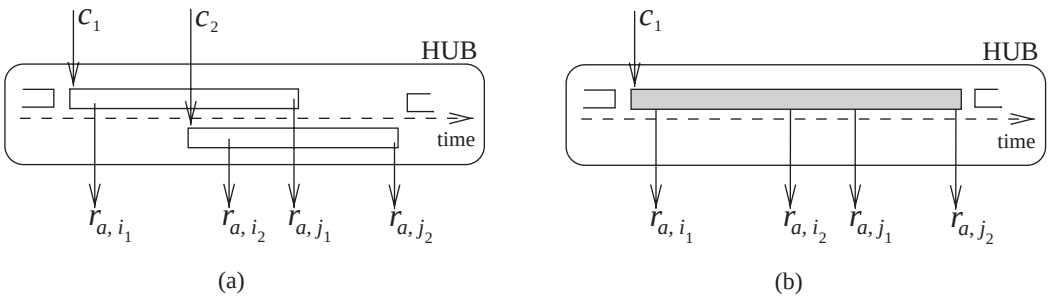

FIGURE 4. The buffer setting assumed in the proof of Lemma 4.1. The buffers in solution O (a) and solution O' (b).

requests can be simulated by an equal number of leaves, each leaf issuing one request).

Let c be a connection into the hub node. A non-empty set of requests gets its data from the buffer associated with connection c. We say that the requests are satisfied by connection c and that connection c is devoted to them. As the following lemma suggests, each connection is devoted to consecutive (with respect to their start times) requests.

LEMMA 4.1. *Consider a feasible allocation problem* $A = (R, G_{Ch}, S)$ *on the chandelier network* G_{Ch}. *Then, for each of the minimum total memory, minimum maximum memory per node and minimum total traffic versions of the allocation problem A, there exists an optimal solution with the property that each connection into the hub is devoted to consecutive (with respect to their start times) requests.*

Proof. We focus on an arbitrary clip a. Assume, for the sake of contradiction, that in an optimal solution O there is always at least one connection devoted to a set of non-consecutive requests. Consider the connection, say c_1, into the hub with the earliest start time which is devoted to a set R' of non-consecutive requests (Figure 4). Let Q be the set of requests for clip a of minimum cardinality which, when added to set R', forms a set of consecutive requests. Let r_{a,i_1} and r_{a,j_1} be the requests in R' with the earliest and latest start times respectively. Let r_{a,i_2} be the request in Q with the earliest start time and let c_2 be the connection that satisfies it. Finally, let R'' be the set of requests satisfied by c_2 and, out of them, r_{a,j_2} be the request with the latest start time. It holds that

$$s(r_{a,i_1}) < s(r_{a,i_2}) < s(r_{a,j_1}) \tag{1}$$

The buffer associated with connection c_1 receives its first frame at time $\text{start}(c_1) \leq s(r_{a,i_1})$ and can buffer $s(r_{a,j_1}) - \text{start}(c_1)$ minutes of the video clip, while the buffer associated with connection c_2 receives its first frame at time $\text{start}(c_2) \leq s(r_{a,i_2})$ and is $s(r_{a,j_2}) - \text{start}(c_2)$ minutes long. Equation (1) implies that the two buffers 'overlap', i.e. there are frames that are present at both buffers simultaneously.

Consider the new solution O' obtained from O by extending the length of the buffer associated with c_1 so that all requests satisfied previously by c_1 and c_2 are now satisfied by c_1 alone. The new length of c_1's buffer is $\max(s(r_{a,j_1}), s(r_{a,j_2})) - \text{start}(c_1)$, which is smaller than or equal to the combined length of the two buffers in the original solution. The equality holds in the case where connection c_2 was devoted to a single request and the buffer associated with c_2 was of zero length.

If O were an optimal solution for the minimum total traffic assignment problem, then O' would be a solution which uses fewer connections than O, which is a contradiction. If O were an optimal solution for the minimum total memory assignment problem, then O' would be a solution that uses buffer area less than or equal to that used by O. If O' uses less memory than O we arrive at a contradiction since we assumed that O was optimal. In the event that O' uses the same memory as O, observe that repeated application of the process described in the proof will result in a solution in which all connections are devoted to sets of consecutive requests. To complete the proof, note that for the case of the chandelier where there is only one node available for buffering, the problem of minimizing the maximum memory per node is equivalent to the problem of minimizing the total memory. □

REMARK 4.1. *In the proof of Lemma* 4.1, *we assumed that in an optimal solution it might hold that the start time of a connection is smaller than the start time of the earliest request it is devoted to (in Figure* 4*a,* $\text{start}(c_2) < s(r_{a,i_2})$*). For the case of the chandelier it is easy to show that the equality always holds. Lemma* 4.1 *actually holds for any node of an arbitrary network and not only for the hub node of the chandelier. As will become evident in Section* 6 *(Figure* 7*), there exist optimal solutions to the assignment problem of minimizing the maximum memory per node on a general graph for which the inequality always holds. In that case, a buffer is 'distributed' over several nodes.*

REMARK 4.2. *Lemma* 4.1 *states that 'connections are devoted to consecutive requests'. As is evident from its proof, it is equivalent to state that 'buffers located at the same node do not overlap', or that 'the same video frame does not appear at more than one buffer at the same node, simultaneously'.*

We now present Algorithm *Chandelier_min_traffic()* which solves the traffic minimization problem on the chandelier. In the event that the algorithm fails to return a solution, the problem is not feasible. The input to the algorithm is a set of requests R partitioned into p disjoint ordered subsets, each corresponding to a different video clip (recall the notation at the end of the previous section).

Observe that we have to devote at least one connection for each video clip. In the event that the available bandwidth out of the server (measured in connections) is smaller than the number p of clips, i.e. $B_{(S,v_1)} < p$, the problem is obviously not feasible. The algorithm ignores this trivial case. If we devote a single connection c_a to each clip a, $1 \leq a \leq p$, all requests in R_a are serviced out of the same buffer. The buffer associated with clip a is of size $s(r_{a,|R_a|}) - s(r_{a,1})$ stretching from the start time of the earliest request $r_{a,1}$ to the start time of the latest request $r_{a,|R_a|}$. Then, $\sum_{a=1}^{p} \left(s(r_{a,|R_a|}) - s(r_{a,1})\right)$ is an upper bound on the total memory used by any feasible solution.

Consider a connection c' which is devoted to the set of requests $r_{a,i}, \ldots, r_{a,j-1}, r_{a,j}, \ldots, r_{a,k}$. If one extra unit of bandwidth becomes available, we can use it to establish a new connection c'' servicing requests $r_{a,j}, \ldots, r_{a,k}$. The buffer used by c' shrinks to size $s(r_{a,j-1}) - s(r_{a,i})$ while the new buffer fed by connection c'' is of size $s(r_{a,k}) - s(r_{a,j})$, resulting in savings in memory equal to the size of the time-gap $\text{gap}(r_{a,j}) = s(r_{a,j}) - s(r_{a,j-1})$ between the starting time requests $r_{a,j}$ and $r_{a,j-1}$. Of course, selecting the request which has the largest time-gap will maximize the savings in memory.

Algorithm *Chandelier_min_traffic*(R, G_{Ch})
Input: Set of requests R and chandelier graph G_{Ch} with edge and node weights.

1. $used\_mem = \sum_{a=1}^{p} \left(s(r_{a,|R_a|}) - s(r_{a,1})\right)$
2. $used\_bw = p$
3. $Solution = \{r_{a,1} \mid 1 \leq a \leq p\}$
4. $Gaps = \{\}$
5. **For** each video clip a, $1 \leq a \leq p$ **do**
   For each request $r_{a,j} \in R_a$, $1 < j \leq |R_a|$ **do**
   $Gaps = Gaps \cup \left\{(gap(r_{a,j}), a, j)\right\}$
6. **While** ($used\_mem > m_{v_1}$ **and** $used\_bw < B_{v_1}$) **do**
   a. Let $g = (gap, a, j)$ be the triplet (corresponding to $r_{a,j}$) with the largest gap-value in *Gaps*.
   b. $Gaps = Gaps \backslash g$
   c. $Solution = Solution \cup \{r_{a,j}\}$
   d. $used\_mem = used\_mem - gap$
   e. $used\_bw = used\_bw + 1$
7. **If** ($used\_mem > m_{v_1}$) **return** ("*The problem is not feasible*")
   else return (*Solution*)

Algorithm *Chandelier_min_traffic()* returns a set of requests. One connection will be established for each of these requests. Based on the fact that there is always an optimal solution in which each connection is devoted to consecutive requests (Lemma 4.1) we can deduce the set of requests that will be serviced by any particular connection, and thus, the buffer size of the connection. This interpretation of Solution into a stream for each request, can be trivially performed in linear to the number of requests time.

LEMMA 4.2. *Consider an assignment problem* $A = (R, G_{Ch}, S)$ *on a chandelier* G_{Ch} *with bandwidth* $B_{(s,v_1)} = i$ *and let* p *be the number of different video clips. Let* Y *be the set of the* $i - p$ *requests of larger gap. Then, the minimum buffer space required to service all requests in* R *is*

$$\sum_{a=1}^{p} \left(s(r_{a,|R_a|}) - s(r_{a,1})\right) - \sum_{r \in Y} gap(r) \qquad (2)$$

Proof. By induction on bandwidth i. For the basis, consider the case where $i = p$. Then, $i - p = 0$ and set Y is empty. Thus, the second summation term is equal to zero. The only way to satisfy all requests is to devote a connection to each video clip and to set the start time of the connection for clip a to the start time of request $r_{a,1}$. Then, the required buffer size for servicing all requests is given by Equation (2).

For the induction step assume that the lemma holds when the bandwidth is equal to $j > p$ connections. Let r be the request with the $(j+1)$th largest gap. By introducing a new connection with start time equal to $s(r)$ we can save $\text{gap}(r)$ units of memory and, thus, get a solution of buffer size given by Equation (2).

To complete the proof for the induction step, we have to show that there does not exist a solution which uses less memory. Given that in an optimal solution the buffers 'do not overlap' (alternative statement of Lemma 4.1), the optimal solution can be described in terms of the gaps and the requests which define them. Given that the sum of j elements (gap-sizes) out of a set is maximized when we add the j largest elements, it is obvious that we cannot get a solution which uses less memory than the quantity stated in Equation (2). □

THEOREM 4.1. *Let* $A = (R, G_{Ch}, S)$ *be an allocation problem on the chandelier network. Then, in* $O(|R| \log |R|)$ *time we can determine whether* A *is feasible and, if it is, we can obtain solutions for the minimum total memory and for the minimum total traffic assignment problems.*

Proof. We use Algorithm *Chandelier_min_traffic()* to obtain a solution to all variants of the problem. Algorithm *Chandelier_min_traffic()* starts by allocating one connection to each different video clip. If the required buffer area (denoted by *used_memory*) is more than that of the available memory, it introduces one additional connection. The algorithm stops introducing new connections when the required buffer area is sufficient to serve all requests or when the bandwidth out of the server is exhausted. In the former case, the problem is feasible and, since the solution returned by the algorithm uses memory equal in size to that suggested by Lemma 4.2, it is a solution to the minimum traffic allocation problem. In the latter case, the problem is obviously not feasible.

Given that for any request r, $\text{gap}(r)$ is positive, Lemma 4.2 implies that the solution to the minimum total memory allocation problem uses all available bandwidth out of the server. Thus, to produce a solution of minimum total memory it is enough to modify the condition at Step 6 of Algorithm *Chandelier_min_traffic()* to become '$used\_bw < B_{v_1}$ **and** $Gaps \neq \emptyset$'. We refer to this modified algorithm as Algorithm *Chandelier_total_memory()*.

Algorithm *Chandelier_min_traffic()* can be easily implemented in $O(|R| \log |R|)$ time by using a priority queue (implemented as a heap) for set Gaps. □

5. RESOURCE ALLOCATION ON THE BROOM TOPOLOGY

In our study of allocation problems on the broom topology (Figure 3b) we again assume that each leaf issues one request, that the start times of the requests for the same video clip are distinct, and that the bandwidth into a leaf is equal to 1.

In trying to identify an optimal solution to a particular assignment problem, it is useful to know that there exists an optimal solution satisfying some properties. This effectively reduces the search space for the optimal solution. The following lemma describes such a property and is valid for any of the optimization problems defined in Section 3 (minimum total memory, minimum maximum memory per node and minimum total traffic). First, we need to define the notion of the bandwidth-usage sequence. Recall that node v_l is the hub-node, and that by Into(v) we denote the set of connections into node v. Given a solution to an allocation problem, the sequence of l integers ($|\text{Into}(v_1)|, |\text{Into}(v_2)|, \ldots, |\text{Into}(v_l)|$) (implied by the solution) is referred as the bandwidth-usage sequence. The sequence $\left(B_{(S,v_1)}, B_{(v_1,v_2)}, \ldots, B_{(v_{l-1},v_l)}\right)$ is referred to as the bandwidth-capacity sequence. A sequence $(x_1, \ldots, x_i, \ldots, x_n)$ is said to be strictly increasing if and only if $x_i < x_{i+1},\ 1 \leq i < n$, and increasing if and only if $x_i \leq x_{i+1},\ 1 \leq i < n$. Strictly decreasing and decreasing sequences are defined similarly.

LEMMA 5.1. *Consider a feasible allocation problem $A = (R, G_{Br}, S)$ on the broom network G_{Br}. Then, for each of the minimum total memory, minimum maximum memory per node, and minimum total traffic versions of allocation problem A, there exists an optimal solution in which the implied bandwidth-usage sequence is increasing.*

Proof. Consider an optimal solution to any version (minimum total memory, minimum–maximum memory per node or minimum total traffic) of the feasible allocation problem A. If the bandwidth-usage sequence is increasing, we are done. If it is not increasing, identify the node v closer to the server which causes the bandwidth-usage sequence to be non-increasing. Let a' be the number of connections into v (out of a possible a) and b' be the number of connections out of v (out of a possible b), where $a' > b'$. For each of the outgoing connections, we can identify its parent connection which feeds it with data (possibly through a buffer). In this way, we can identify up to b' distinct connections into v which are useful in the sense that the data they carry is retransmitted. The remaining incoming connections can be safely dropped, producing an optimal solution in which the number of connections into node v is equal to the connections out of v. We continue this process until we cannot identify any node v which causes the bandwidth-usage sequence to be non-increasing, leaving us with an increasing sequence. The process will obviously terminate since each node on the broom-stick has at least one connection into it. □

REMARK 5.1. *Observe that Lemma* 5.1 *also holds for the bandwidth-usage of an individual video clip. The proof is identical.*

Without loss of generality, we can focus only on problems with increasing bandwidth-capacity sequence. Lemma 5.1 established that having greater incoming than outgoing bandwidth for any node does not help in obtaining a better solution. With a process identical to that described in the proof of Lemma 5.1 we can always obtain an equivalent increasing bandwidth capacity sequence which we can use (instead of the original non-increasing one given in the definition of the broom G_{Br}) to solve the allocation problem. The elements of any decreasing sub-sequence can be replaced by the smallest element, and we repeat until no decreasing sub-sequence exists. For example, the bandwidth-capacity sequence (7, 6, 4, 7, 8, 3, 5) is equivalent to (3, 3, 3, 3, 3, 3, 5) as it is established by the sequence of transformations ($\underline{7, 6, 4}$, 7, 8, 3, 5) → (4, 4, 4, 7, $\underline{8, 3}$, 5) → (4, 4, 4, $\underline{7, 3, 3}$, 5) → ($\underline{4, 4, 4, 3, 3, 3}$, 5) → (3, 3, 3, 3, 3, 3, 5).

5.1. Minimum total memory allocation on G^{∞}_{Br}

We now focus on the minimum total memory allocation problem for the broom network. We concentrate on the case where there are no memory bounds on the amount of memory that we can use for buffering at any node of the broom. To emphasize this assumption, we denote the broom network by G^{∞}_{Br}. Let $A = (R, G^{\infty}_{Br}, S)$ be the allocation problem, and let b be the minimum bandwidth capacity on the broom-stick of G^{∞}_{Br}. We further assume that the number of different clips in R is smaller than or equal to b, otherwise, the problem is trivially non-feasible. Given this assumption and the fact that in G^{∞}_{Br} the amount of available memory per node is unbounded, allocation problem A is always feasible.

The following lemmas will be used in obtaining an optimal solution to the problem.

LEMMA 5.2. *Consider a feasible allocation problem $A = (R, G^{\infty}_{Br}, S)$ on the broom network G^{∞}_{Br}. Then, there exists an optimal solution to the minimum total memory allocation problem A with the property that all incoming connections are retransmitted immediately, i.e. for each connection c_1 into an arbitrary node v there exists a connection c_2 out of v with* $\text{start}(c_1) = \text{start}(c_2)$ *and* $\text{clip}(c_1) = \text{clip}(c_2)$.

Proof. Consider an optimal solution. If all connections are retransmitted immediately we are done. Otherwise, we can transform it to an equivalent solution which satisfies the property. Identify a connection c_1 entering node v such that the connection with the earliest start time out of all connections which get data from buf(c_1), say c_2, has $\text{start}(c_2) > \text{start}(c_1)$ (Figure 5a). Obviously, $\text{clip}(c_1) = \text{clip}(c_2)$. We can get a solution that uses exactly the same number of connections and at most the same total buffer space as the original one by extending the buffer of the parent connection of c_1 to stretch up to time $\text{start}(c_2)$ and by updating the start time of connection c_1 to be equal to that of c_2 (Figure 5b). Repeating the process will yield an optimal solution in which all connections are immediately retransmitted. □

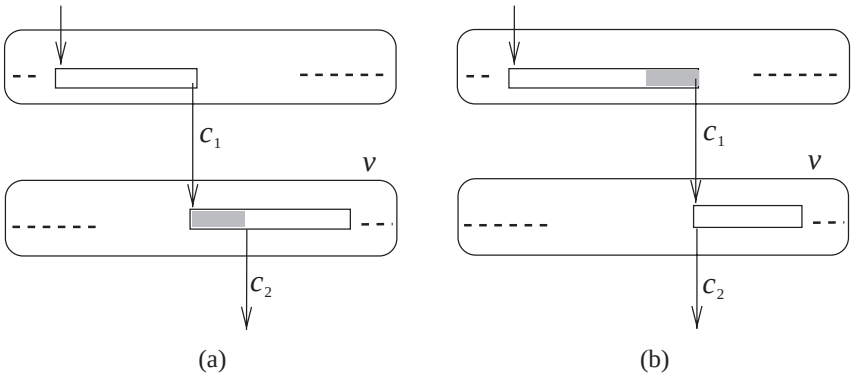

FIGURE 5. Incoming connections into a node are immediately retransmitted. (a) Delayed retransmission and (b) immediate retransmission.

REMARK 5.2. *Note that the proof fails in the case where the available memory at each node is bounded. Therefore, our focus is on network G^{∞}_{Br} instead of G_{Br}.*

REMARK 5.3. *The lemma also holds for the problem of minimum total traffic (identical proof). However, it does not hold for the problem of minimum maximum memory per node where a setting such as the one shown in Figure* 5*a is desirable when an optimal solution spreads a large buffer over a group of consecutive nodes on the broom-stick.*

LEMMA 5.3. *Consider a feasible allocation problem $A = (R, G_{Br}, S)$ and let b be the minimum bandwidth capacity on the path from the server to the hub on broom G_{Br}. Then, there exists an optimal solution to the minimum total memory allocation problem A with exactly b connections out of the server.*

Proof. Lemma 5.1 implies that there exists an optimal solution for which the number of connections out of the server is at most b. We show that it is exactly b. We assume that for the problem at hand the number of requests is greater than b, otherwise the problem can be solved trivially without using any buffering area. Assume that we have an optimal solution which uses fewer than b connections out of the server. Let v be the node closer to the server that hosts a buffer and let c be the last connection out of a buffer located at v. The facts that b is the minimum bandwidth capacity and that the number of connections out of the server is smaller than b imply that we can establish one new connection for each edge at the path from the server to node v. Let each of these new connections have start time equal to start(c) and carry the same clip clip(c). By setting the parent connection of c to be, out of these new connections, the one that enters v, we can reduce the size of the buffer at v, leading to a solution with less memory than the minimum, a clear contradiction. □

LEMMA 5.4. *Consider a feasible allocation problem $A = (R, G^{\infty}_{Br}, S)$ on broom G^{∞}_{Br}. Then, there exists an optimal solution to the minimum total memory allocation problem A with (i) the same number of connections for every edge on the broom-stick, and (ii) all the buffers located at the hub.*

Proof. Consider an optimal solution of increasing bandwidth usage with exactly q connections out of the server, which also satisfies the property that each connection is retransmitted immediately. Such a solution always exists (Lemmas 5.1 and 5.2). We will show that we can transform it into a solution that uses the same amount of memory and exactly q connections for every edge on the broom-stick. Consider the node on the broom-stick that is closer to the server, say v, with q incoming and $q+k,\ k>0$ outgoing connections. (If such a node does not exist there is nothing to prove.) Then, node v has at least one buffer located to it (Figure 6).

Consider the leftmost (in time) such buffer and assume that d connections get their data from it, where $d \leq k$. By moving the buffer to the node below v (and possibly attaching it to a buffer at that node) we do also get a solution that uses the same amount of memory and has $m+k'$ connections out of v, where $k' = k-d$. By repeating the procedure, we can get an equivalent solution where the number of connections out of node v is equal to q, and thus pushing the node which violates the property further away from the server. By repeating, we reach the situation where the only node with more than q connections out of it is the hub node. Observe that at the end of the transformation all buffering takes place at the hub. □

We can now state Algorithm *Broom_total_memory()* which computes an optimal solution for the minimum total memory allocation problem $A = (R, G^{\infty}_{Br}, S)$ on broom G^{∞}_{Br}.

Algorithm *Broom_total_memory*(R, G^{∞}_{Br})
Input: Set of requests R and broom graph G^{∞}_{Br} with edge and node weights.

1. Let b be the minimum bandwidth capacity on the broom-stick of G^{∞}_{Br}.
2. Let G^{∞}_{Ch} be the chandelier graph obtained by collapsing the broom-stick of G^{∞}_{Br} into a single edge of bandwidth capacity $B_{(S,hub)} = b$.
3. Call Algorithm *Chandelier_total_memory*(R, G^{∞}_{Ch}) [defined in proof of Theorem 4.1]
4. Expand the solution for G^{∞}_{Ch} to one for G^{∞}_{Br}.

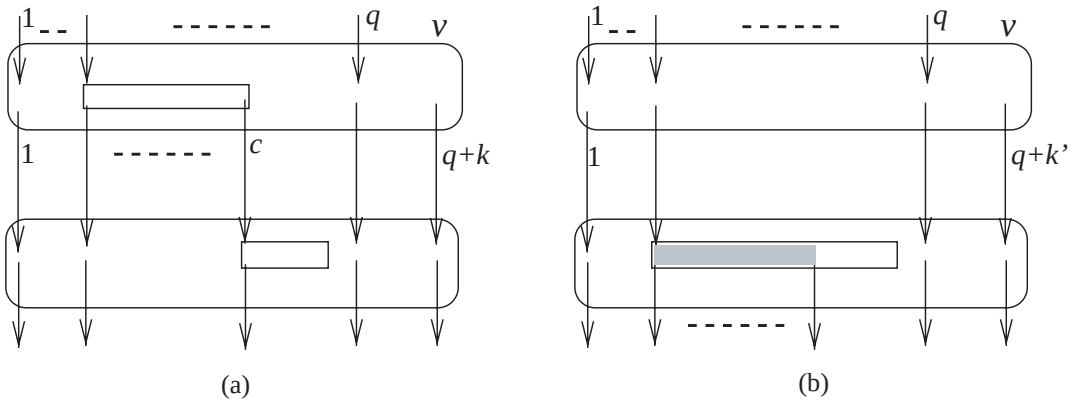

FIGURE 6. The number of connections into node v is reduced by pushing the buffering area closer to the hub. (a) Buffers are located at node v and (b) the buffers were pushed below node v.

Step 4 of Algorithm *Broom_total_memory()* can be easily implemented. Let the server node S also be denoted by v_0 and recall that the length of the broom-stick is l. For every connection c in the solution of (R, G^{∞}_{Ch}, S), we simply create the set of connections $\{c_i : 1 \leq i \leq l$ **and** $\text{orig}(c_i) = v_{i-1}$ **and** $\text{dest}(c_i) = v_i$ **and** $\text{clip}(c_i) = \text{clip}(c)$ **and** $\text{start}(c_i) = \text{start}(c)\}$.

THEOREM 5.1. *Let $A = (R, G^{\infty}_{Br}, S)$ be an allocation problem on the broom network G^{∞}_{Br}. Then, in $O(|R| \log |R|)$ time we can obtain a solution for the minimum total memory allocation problem.*

Proof. We employ Algorithm *Broom_total_memory()*. The proof of correctness follows from Lemmas 5.3 and 5.4.

Step 3 is the most time-consuming step of the algorithm. It needs $O(|R| \log |R|)$ time to terminate (see Theorem 4.1). Note that the solution is represented in a compact form in the sense that it is enough to describe the connections over only one edge of the broom-stick. If we want to get a solution in the form of one stream per request we need to describe at most $l \cdot b$ connections on the broom-stick plus an extra $|R|$ connections into the leaves. Thus, since we assume that $b < |R|$, we need $O(\max(l, \log |R|) \cdot |R|)$ time to compute them. □

6. THE MINIMUM MAXIMUM MEMORY PER NODE ALLOCATION PROBLEM

6.1. The front-selection problem

Consider a two-level broom (it has exactly one intermediate node between the server and the hub) where the bandwidth out of the server is B and the bandwidth into the hub is $B + k$, $k > 0$. As shown in Section 5.1, there exists an optimal solution for the minimum total memory problem in which exactly B connections are used for each edge, each connection is retransmitted immediately, and all buffering area is located at the hub. In such an optimal solution, we can reduce the amount of memory located at the hub by selecting portions at the front of the buffers which satisfy at most k requests and moving them to the node above the hub, thus taking advantage of the extra available bandwidth into the hub. This is illustrated in Figure 7 where, for the set of requests with starting times in $\{0, 5, 12, 22, 26, 30\}$, the solution computed in Section 5.1 which minimizes the total memory uses a buffering area of 20 min, all located at the hub (Figure 7a). A solution which uses two extra connections into the hub 'lifts' 9 min of memory to the node above the hub and, as a result, reduces the maximum memory per node to 11 (still at the hub) and at the same time leaves the total memory used unchanged (Figure 7b).

The solution of Figure 7b has the property that the data over each connection entering the hub are retransmitted immediately. If we relax this restriction we can get the solution of Figure 7c where the same amount of buffering area (10 min) is used at both the hub and the node above it. This improved solution is possible only because we allow the buffering area (of 7 min) required to serve the request which starts at time '12-minutes' to spread over two nodes, 2 min of buffering area at the hub and 5 min at the node above it. This balances the total amount of buffering area between the two nodes and gives a solution to the MMMN problem for the broom network and the set of requests used in the example of Figure 7.

Observe that the total amount of memory remains unchanged only if we lift to the node above the hub section(s) from the 'front' of the buffer(s). We cannot move a section from the middle of some buffer since we cannot feed it with data from the existing connections out of the server (their number is assumed to remain unchanged).

In the optimal solution to the MMMN problem which is presented in Figure 7c, only one connection is not retransmitted immediately. As the following lemma suggests, a feasible allocation problem always has an optimal solution to the MMMN problem where at most one of the connections between two adjacent nodes is not retransmitted immediately.

LEMMA 6.1. *Consider a feasible allocation problem $A = (R, G, S)$ and let $e = (v, w)$ be an edge of network G. Then, there exists a set of connections $C(A)$ which is a solution to the MMMN allocation problem with the property that at*

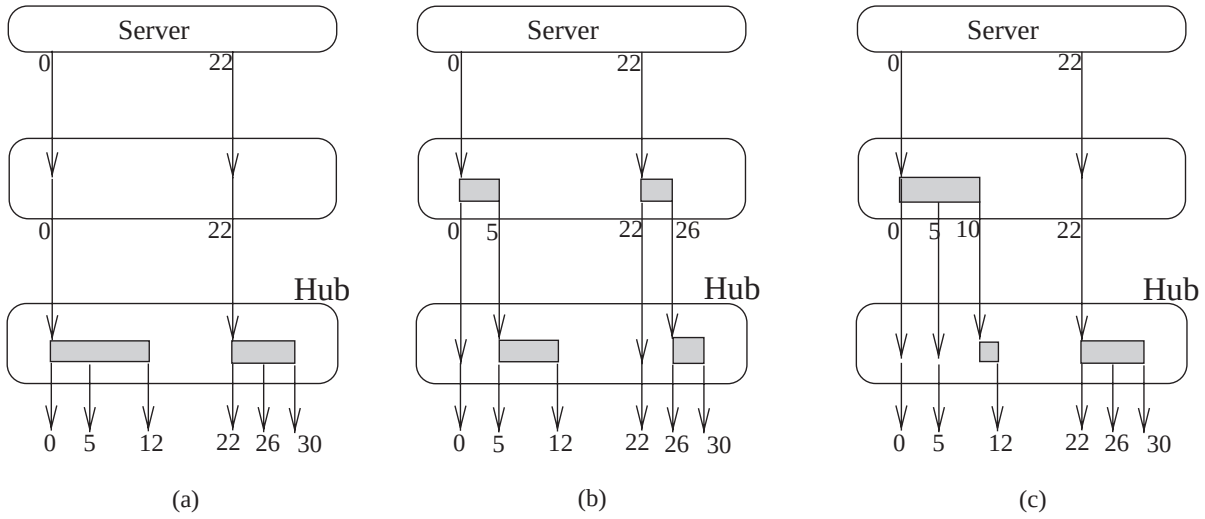

FIGURE 7. Use of two extra connections into the hub reduces the amount of memory per node without increasing the total memory. (a) All buffers are at the hub, (b) buffers were pushed above the hub and immediate retransmission required and (c) balanced distribution of buffers. Only possible without immediate retransmission.

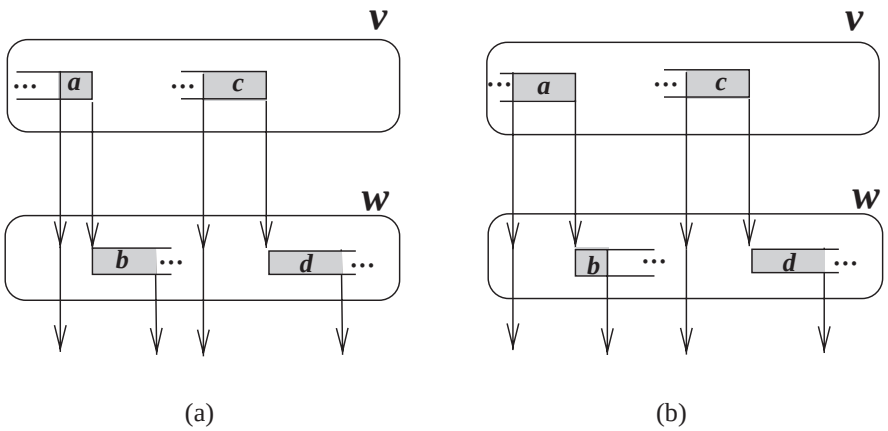

FIGURE 8. One of the two non-immediately transmitted connections can always be eliminated (proof of Lemma 6.1). (a) The smallest segment is at node v and (b) the smallest segment is at node w.

most one of the connections entering node w over edge e is not retransmitted immediately.

Proof. Consider solution $C(A)$. If for every edge $e = (v, w)$ it holds that at most one connection in solution $C(A)$ enters node w over edge e, we are done. If this is not the case, let $e = (v, w)$ be an edge for which the property does not hold. We will show how to get an equivalent solution $C'(A)$, i.e. a solution which uses at most the same number of connections over edge e and leaves the amount of buffering area at each of the nodes v and w unchanged.

Consider two arbitrary connections entering node w over edge $e = (v, w)$ which are not immediately retransmitted. These two connections define four segments of buffering area, two at node v (segments a and c in Figure 8) and two at node w (segments b and d in Figure 8). We consider two cases based on the location of the minimum-size segment.

CASE 1. *The segment of minimum size is at node v.*

Without loss of generality let it be segment a (see Figure 8a). We can get an equivalent solution with one less non-immediately transmitted connection by:

(i) moving segment a to node w (this also saves one connection), and,
(ii) moving up to node v from the front of segment d an amount of buffering area equal to the size of segment a. This is possible since we assumed that segment a is of minimum size. The resulting (from the above transformations) set of connections $C'(A)$ is a solution to problem A which leaves the buffering area at each node unchanged. In addition, its use of bandwidth is improved since it uses one less connection over edge e.

CASE 2. *The segment of minimum size is at node w.*

Without loss of generality let it be segment b (see Figure 8b). We can get an equivalent solution with one less non-immediately transmitted connection by:

(i) moving segment b up to node v, and,
(ii) moving down to node w from the rear of segment c an amount of buffering area equal to the size of segment b. This is possible since we assumed that segment a is of minimum size. The resulting (from the above transformations) set of connections $C'(A)$ is also a solution to problem A which leaves the buffering area at each node unchanged. □

The above discussion leads to the formulation of the following problem.

6.1.1. Front-selection

Instance: We are given a positive integer k, n sequences of positive integers $S^i = (s_1^i, s_2^i, \ldots, s_{k_i}^i)$, $1 \leq i \leq n$, $1 \leq k_i \leq k$, and an integer λ such that $0 \leq \lambda \leq \max_{\substack{i=1\cdots n \\ j=1\cdots k_i}} \{s_j^i\}$.

Question: Is there a way to select at most k elements from the front of the sequences such that their sum exceeds the sum of the remaining elements by an even number which is at most 2λ? (Since $0 \leq \lambda$, the difference might be equal to zero.)

More formally, is there a sequence of integers $X = (x_1, x_2, \ldots, x_n)$ such that the following hold:

$$0 \leq x_i \leq k_i \quad \text{for } 1 \leq i \leq n, \tag{3}$$

$$\sum_{i=1}^{n} x_i \leq k, \tag{4}$$

$$\left(\sum_{i=1}^{n}\sum_{j=1}^{x_i} s_j^i\right) - \left(\sum_{i=1}^{n}\sum_{j=x_{i+1}}^{k_i} s_j^i\right) \text{ is even}, \tag{5}$$

and

$$0 \leq \left(\sum_{i=1}^{n}\sum_{j=1}^{x_i} s_j^i\right) - \left(\sum_{i=1}^{n}\sum_{j=x_{i+1}}^{k_i} s_j^i\right) \leq 2\lambda. \tag{6}$$

THEOREM 6.1. *The front-selection problem is NP-complete.*

Proof. By restriction. We show that the partition problem [30, 31], which is known to be NP-complete, is a special case of the front-selection problem.

In an instance of the partition problem we are given a set A of n positive integers, and we are asked whether it is possible to partition it into two disjoint sets A_1 and A_2 such that the sum of the elements in A_1 is equal to the sum of the elements in A_2.

Consider the case of the front-selection problem where each of the n sequences consists of a single integer, $k = n$ and $\lambda = 0$. The set of the n single-element sequences can be treated simply as a set of n elements (positive integers). Then, sequence X (of the front-selection problem) defines a partition of the n-element set into two disjoint sets such that the sum of the elements of the two sets are equal. Thus, the partition problem is a special case of the front-selection problem. □

6.2. On the computational complexity of the MMMN allocation problem

Our aim is to prove that a natural version of the allocation problem of minimizing the maximum memory per node on a general graph is hard. In doing so, we consider the following decision problem on a two-level broom:

6.2.1. MMMN-opt-total (Minimum maximum memory per node with respect to the optimal total memory)

Instance: Set of requests $R = \{r_1, r_2, \ldots, r_d\}$ for the same clip such that $\text{start}(r_i) < \text{start}(r_{i+1})$, $1 \leq i < d$, 2-level broom $G_{Br_2}^{\infty}$, and positive integer M_{Node}.

Question: Does there exist a way of satisfying all requests in R such that the maximum memory per node is smaller than or equal to M_{Node} and the total memory used is equal to that of an optimal solution to the minimum total memory problem for request set R on $G_{Br_2}^{\infty}$?

THEOREM 6.2. *The MMMN-opt-total problem is NP-complete.*

Proof. We reduce the front selection problem to the MMMN-opt-total problem. Consider an instance $I_{\text{FS}} = (S^1, S^2, \ldots, S^n, k, \lambda)$ of the front selection problem and let $g = 1 + \max\{s_j^i : s_j^i \in S^i,\ 1 \leq j \leq k_i,\ 1 \leq i \leq n\}$. Based on I_{FS}, we construct in polynomial time an instance $I_{\text{MMMN-ot}} = (R, G_{Br_2}^{\infty}, M_{\text{Node}})$ of the MMMN-opt-total problem.

Let the set of requests be $R = \{r_j^i : 1 \leq i \leq n,\ 0 \leq j \leq k_i\}$ where

$$\text{start}(r_j^i) = \sum_{a=1}^{i-1}\left(g + \sum_{b=1}^{k_i} s_b^a\right) + \sum_{b=1}^{j} s_b^i \tag{7}$$

The two-level broom $G_{Br_2}^{\infty}$ consists of the server S, the hub v_2, the intermediate node v_1 between the server and the hub, and $\sum_{i=1}^{n} k_i + n$ leaves hanging from the hub. The bandwidth $B_{(S,v_1)}$ out of the server is equal to n, the bandwidth $B_{(v_1,v_2)}$ into the hub is equal to $n + k$, and the bandwidth into each leaf is equal to 1. Each leaf issues a single request out of those in R.

Finally, we set $M_{\text{Node}} = \text{total}/2$ where total is the amount of memory used in an optimal solution to the minimum total memory allocation problem $(R, G_{Br_2}^{\infty}, S)$ (which can be computed in polynomial time by Algorithm *Broom_total_memory()*).

The start times in R can be interpreted as follows: consider the sequence obtained by attaching integer g to the end of the first $n - 1$ original sequences, and, then, by concatenating all the sequences into a new one. The new sequence has $\sum_{i=1}^{n} k_i + n - 1$ elements and is separated by the element with value g into the n original sequences. Treat this sequence as the sequence of differences ('gaps' in the terminology of

the presented algorithms) in start times between $\sum_{i=1}^{n} k_i + n$ consecutive requests, where the starting time of the first request is equal to zero. Then, the start times are given by Equation (7). In the rest of the proof, we say that requests $\{r_j^i : 0 \leq j \leq k_i\}$ correspond to sequence S^i, $1 \leq i \leq n$ and vice versa.

Consider the solution produced by Algorithm *Broom_total_memory()* for the minimum total memory allocation problem $(R, G^{\infty}_{Br_2}, S)$. It uses n connections on every edge on the broomstick and, after saving space for the $n-1$ 'large' time differences of size g, it creates n buffers at the hub, where the set of requests which are serviced by the ith buffer is $\{r_j^i : 0 \leq j \leq k_i\}$ and corresponds to sequence S^i, $1 \leq i \leq n$ (Lemmas 5.3 and 5.4). Then, the total memory used is total $= \sum_{i=1}^{n} \sum_{j=1}^{k_i} s_j^i$.

Assume that instance $I_{\text{MMMN}-\text{ot}}$ has a solution. Since the buffer area is measured in units of memory-minutes (which are indivisible), it follows that total is an even number, and exactly total/2 units of memory are placed on each of the nodes v_2 (the hub) and v_1. Given that the total memory used has to be equal to total, the buffers at v_1 and v_2 do not overlap. So, the solution to the instance of $I_{\text{MMMN}-\text{ot}}$ can be thought to be derived from the solution given by Algorithm *Broom_total_memory()* by lifting to v_1 the buffering area that corresponds to at most k requests. For each of those requests, a new connection was created and at most one of these connections is not immediately retransmitted (Lemma 6.1).

By identifying the connections which feed data to the buffers at the hub, we can partition each of the n groups of requests into a leftmost and a rightmost part (either of which can be empty). This gives a partition of the corresponding sequences. In that partition, the number s (element of some sequence) that corresponds to the single non-immediately retransmitted connection is included to the leftmost part. This implies that the difference of the sums of the numbers at each part (leftmost and rightmost) is even. Moreover, since $s \leq \lambda$, the sum of the elements in the leftmost part is greater than the sum of the elements in the rightmost part by at most 2λ. Thus, based on a solution to instance $I_{\text{MMMN}-\text{ot}}$ of the MMMN-opt-total problem, we can produce a solution to instance I_{FS} of the front-selection problem.

Conversely, assume that instance I_{FS} of the front-selection problem has a solution. Then, by placing at node v_1 the buffer area that corresponds to the elements removed from the sequences, and by placing at the hub the buffer area which corresponds to the remaining elements, we get a solution where the buffer area at node v_1 is greater than that of the hub by an even number of at most 2λ units (minutes) of memory. Thus, by moving at most λ units of memory from node v_1 to the hub we can get a solution where at each node we use exactly total/2 buffering area, as required by a solution to instance $I_{\text{MMMN}-\text{ot}}$. Note that one of the streams in the solution might not be immediately retransmitted. □

We now define the minimum maximum memory per node with respect to total memory usage (MMMN-total) and we state the main result of this section.

6.2.2. *MMMN-total (minimum maximum memory per node with respect to total memory usage)*

Instance: An allocation problem $A = (R, G, S)$, and positive integers M_{Node} and M_{total}.

Question: Does there exist a solution to problem A such that the maximum memory per node is smaller than or equal to M_{Node} and the total memory used is smaller than or equal to M_{total}?

THEOREM 6.3. *The MMMN-total problem is NP-complete.*

Proof. Simply observe that the MMMN-opt-total problem which was shown to be NP-complete in Theorem 6.2 is a special case of the MMMN-total problem where the graph G is restricted to a two-level broom $G^{\infty}_{br_2}$, the set of requests is restricted to the same clip, and M_{total} is set to be equal to the optimal solution of the minimum total memory problem for the same set of requests on broom $G^{\infty}_{br_2}$. □

7. RESOURCE ALLOCATION ON A GENERAL TREE

In this section we present a heuristic for computing a solution to an allocation problem on a general tree where the root of the tree acts as the server and all requests are issued at the leaves. Without loss of generality, we assume that the root has only one child. If this is not the case, we can solve the problem on each tree obtained by deleting all but one subtree of the root and, at the end, by combining the solutions to these sub-problems. We assume that there are no bounds on the amount of memory that can be used for buffering at any node of the tree and, to emphasize this, we denote the tree by T^{∞}.

Heuristic *Tree_allocation*(R, T^{∞})
Input: Tree T^{∞} with edge weights only, and set of requests R (all issued at the leaves of T^{∞})

1. $Solution = \{\}$
2. **while** (T^{∞} contains at least one internal node different than the root) **do**

   a. Let v be an internal node that has only leaves as its children.
   Denote by T_v^{∞} the subtree rooted at v.
   Let $R_v \subseteq R$ be the set of requests issued by the children of v.
   b. Let chandelier Ch_v^{∞} be formed by the parent $p(v)$ of v (root), v itself (hub), and the children of v. The bandwidth into v is equal to its original value $B_{(p(v),v)}$.
   c. Call Algorithm *Chandelier_total_memory()*. Let the solution place b buffers at v and let $C_v = \{c_1, c_2, \ldots, c_b\}$ be the set of connections into v which feed these buffers with data and C_{leaves} be the set of connections into the children of v.
   d. $Solution = Solution \cup C_{leaves}$
   e. Update T^{∞} by replacing T_v^{∞} with b nodes $v_1, v_2, \ldots, v_b$, each with one unit of bandwidth into it.

f. **If** (T^{∞} contains at least one internal node different than the root)
then Update R by:
i) removing the requests issued by the children of v and then,
ii) inserting b new requests $\{r_i\colon 1 \leq i \leq b\}$ (each one corresponding to a request in C_v) where $s(r_i) = start(c_i)$, $clip(r_i) = clip(c_i)$, and $i(r_i) = v_i,\ 1 \leq i \leq b$.
else $Solution = Solution \cup C_v$

3. **return** *Solution*

The solution returned by Heuristic *Tree_allocation()* consists of a set of connections that carry all the necessary information we need in order to deduce how to set up the buffers at the nodes of T^{∞} so we can satisfy all requests in R.

Unfortunately, we were not able to provide any non-trivial bound on the quality of the heuristic. We were only able to demonstrate (by constructing simple instances of the allocation problem) that, even for the broom network, it does not compute the optimal solution to either the minimum total memory or minimum maximum memory per node problem. Despite that, the heuristic has been employed in a prototype system and has performed satisfactorily.

8. CONCLUSION

In this paper we examined issues related to the algorithmic support of the delayed-multicast protocol. We introduced the MTM, MTT and the MMMN delayed-multicast allocation problems. We examined these problems on two networks of practical interest, namely, the chandelier and the broom networks. We provided polynomial time algorithms for solving the MTM and the MTT problems on the chandelier network and the MTM problem on the broom network. We also showed that a version of the decision-MMMN problem on a general graph is NP-complete. Finally, we presented a heuristic method for obtaining a solution for the MTM problem on tree networks.

Several open problems which arose from our research need to be addressed, among them the following:

(i) *The complexity of the allocation problems.* The decision MMMN-total problem, which was shown to be NP-complete in Section 6, requires that the total memory used is bounded by a parameter of the problem. Is the problem NP-complete when no bounds exist on the total memory used? What about the MTM, and the MTT problems?

(ii) *Allocation problems on the tree network.* What is the complexity of the allocation problems on the tree network? Section 6 provided evidence towards the conjecture that the MMMN allocation problem is hard, even in the case of a two-level broom. What about the MTM and the MTT problems on tree networks? In Section 7 we developed a heuristic method which constructs a non-optimal solution to an allocation problem on a tree network. Can we provide bounds on the performance of that (or any other) heuristic method? Experimental results on a heuristic which utilizes prior knowledge about each video clip's popularity have been presented in [32].

(iii) *Providing support for a true VoD system.* Design and analyze buffer reallocation algorithms. This is an on-line version of the problem. A reallocation algorithm rearranges the buffers at the nodes of the network in response to the arrival of a new request, or to the dropping of an existing one.

REFERENCES

[1] El-Gindy, H., Nguyen, C. and Symvonis, A. (2000) "Scheduled-Multicast" with application in multimedia networks. In *IEEE Int. Conf. on Networks 2000* (*ICON 2000*), Singapore, September 5–8, pp. 191–198.

[2] Glinos, N., Hoang, D. B., Nguyen, C. and Symvonis, A. (2002) Algorithmic support for video-on-demand based on the delayed-multicast protocol. In *The 9th Int. Colloquium on Structural Information & Communication Complexity* (*SIROCCO 9*), Andros, Greece, June 10–12, pp. 133–148.

[3] Özden, B., Rastogi, R. and Silberschatz, A. (1994) On the storage and retrieval of continuous media data. In *3rd Int. Conf. on Knowledge Management*, Gaithersburg, MD, November 29–December 2, pp. 322–328.

[4] Garofalakis, M., Özden, B. and Silberschatz, A. (1998) On periodic resource scheduling for continuous media databases. *VLDB J.*, **7**(4), 206–225.

[5] Patterson, D. A., Gibson, G. A. and Katz, R. H. (1988) A case for Redundant Arrays of Inexpensive Disks (RAID). In *Proc. Conf. on Management of Data*, Chicago, IL, June 1–3, pp. 109–116.

[6] Kamath, M., Ramamritham, K. and Towsley, D. (1995) Continuous media sharing in multimedia database systems. In *Proc. Fourth Int. Conf. on Database Systems for Advanced Applications* (*DASFAA'95*), Singapore, April 10–13, pp. 79–86. World Scientific Publishing Co. Pte Ltd, Singapore.

[7] Shi, W. and Ghandeharizadeh, S. (1997) Buffer sharing in video-on-demand servers. In *Conf. on Measurement and Modeling of Computer Systems* (*SIGMETRICS'97*), June 15–18, pp. 13–20, Seattle, Washington, DC.

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

[9] Aggarwal, C., Wolf, J. and Yu, P. (1996) A permutation-based pyramid broadcasting scheme for video-on-demand systems. In *IEEE Multimedia Computing and Systems Conf.*, Hiroshima, Japan, June 17–23, pp. 118–126.

[10] Hua, K. A. and Sheu, S. (1997) Skyscrapper broadcasting: a new broadcasting scheme for metropolitan video-on-demand system. In *Proc. ACM SIGCOMM'97*, Cannes, France, September 16–18, pp. 89–100.

[11] Juhn, L. S. and Tseng, L. M. (1997) Fast broadcasting for hot video access. In *Proc. Fourth Int. Workshop on Real-Time Computing Systems and Applications*, Taipei, Taiwan, October, pp. 237–243. IEEE Computer Society Press.

[12] Paris, J. F., Carter, S. W. and Long, D. D. (1999) A hybrid broadcasting protocol for video on demand. In *Proc. Multimedia Computing and Networking Conf.*, San Jose, CA, January 25–27, pp. 317–326.

[13] Tseng, Y. C., Yang, M. H. and Chang, C. H. (2002) A recursive frequency-splitting scheme for broadcasting hot videos in VOD service. *IEEE Trans. Commun.*, **50**(8), 1348–1355.

[14] Tseng, Y. C., Yang, M. H., Hsieh, C. M., Liao, W. H. and Sheu, J. P. (2001) Data broadcasting and seamless channel transition for highly-demanded videos. *IEEE Trans. Commun.*, **49**(5), 863–874.

[15] McCanne, S. (1999) Scalable multimedia communication: using IP multicast and lightweight sessions. *IEEE Internet Comput.*, **3**(2), 33–45.

[16] Aggarwal, C., Wolf, J. and Yu, P. (1996) On optimal batching policies for video-on-demand servers. In *IEEE Multimedia Computing and Systems Conf.*, Hiroshima, Japan, June 17–23, pp. 253–258.

[17] Dan, A., Sitaram, D. and Shahabuddin, P. (1994) Scheduling policies for an on-demand video server with batching. In *ACM Multimedia Conf.*, San Francisco, CA, October 15–20, pp. 15–23.

[18] Krishnan, R. and Little, T. D. C. (1997) Service aggregation through rate adaptation using a single storage format. In *Proc. 7th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video*, St. Louis, MO, May, pp. 273–282.

[19] Basy, P., Narayanan, A., Krishnan, R. and Little, T. D. C. (1998) An implementation of dynamic service aggregation for interactive video delivery. In *Proc. SPIE–Multimedia Computing and Networking*, San Jose, CA, January 26–28, pp. 111–122.

[20] Golubchik, L., Lu, J. C. S. and Muntz, R. (1995) Reducing I/O demand in video-on-demand storage servers. In *Conf. on Measurement and Modeling of Computer Systems* (*SIGMETRICS'95*), Ottawa, Canada, May 15–19, pp. 25–36.

[21] Aggarwal, C., Wolf, J. and Yu, P. S. (1996) On optimal piggyback merging policies for video-on-demand systems. In *Conf. on Measurement and Modeling of Computer Systems* (*SIGMETRICS'96*), Philadelphia, PA, May 23–26, pp. 200–209.

[22] Hua, K. A., Cai, Y. and Sheu, S. (1998) Patching: a multicast technique for true video-on-demand services. In *ACM Multimedia Conf.*, Bristol, September 12–16, pp. 191–200.

[23] Cai, Y., Hua, K. A. and Vu, K. (1999) Optimizing patching performance. In *Proc. SPIE–Multimedia Computing and Networking*, Santa Clara, CA, January 25–27, pp. 204–215.

[24] Sen, S., Gao, L., Rexford, J. and Towsley, D. (1999) Optimal patching schemes for efficient multimedia streaming. In *The 9th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video*, AT&T Learning Center, Basking Ridge, NJ, June 23–25, pp. 32–44.

[25] Eager, D., Vernon, M. and Zahorjan, J. (2000) Bandwidth skimming: a technique for cost-effective video-on-demand. In *Proc. SPIE–Multimedia Computing and Networking*, Santa Jose, CA, January 24–26, pp. 206–215.

[26] Mahanti, A., Eager, D. L., Vernon, M. K. and Sundaram-Stukel, D. (2001) Scalable on-demand media streaming with packet loss recovery. In *Proc. of ACM SIGCOMM'01*, New York, NY, August 27–31, pp. 97–108.

[27] Liao, W. and Li, V. O. K. (1997) The split and merge protocol for interactive video-on-demand. *IEEE Multimedia Mag.*, 51–62.

[28] Chan, S. H. G. and Tobagi, F. (2001) Distributed servers architecture for networked video services. *IEEE/ACM Trans. Networking*, **9**, 125–136.

[29] Waldvogel, M., Deng, W. and Jamakiraman, R. (2003) Efficient buffer management for scalable media-on-demand. In *Proc. SPIE–Multimedia Computing and Networking*, Santa Clara, CA, January 20–24, pp. 192–199.

[30] Garey, M. R. and Johnson, D. S. (1979) *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.

[31] Karp, R. M. (1972) Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W. (eds), *Complexity and Computer Computations*, pp. 85–103. Plenum Press, New York.

[32] Nguyen, C., Hoang, D. B. and Symvonis, A. (2003) Multi-level caching with delayed-multicast for video-on-demand. To appear in the *Proc. 7th IASTED Int. Conf. on Internet and Multimedia Systems and Applications (IMSA'03)*, Hawaii, August 13–15, pp. 699–705.