Resource Allocation for Video-on-Demand with ``Delayed-Multicast'' Protocol

Chi Nguyen, Doan B. Hoang
Department of Computer Systems
University of Technology Sydney
Sydney, Australia.
{chi,dhoang}@it.uts.edu.au

Antonios Symvonis
Department of Mathematics
University of Ioannina,I oannina,G reece.
symvonis@cc.uoi.gr

Abstract-**"Delayed-Multicast" is a novel transmission technique which uses internal nodes in the transmission paths from the server to the clients to buffer data streams. The buffers areu sed to service later requests without having to start new streams from the server,t hus bringing the benefits of traditional multicast without the constraint that all requests must be serviced at the same time.**

In this paper, we describe our new scalable "Delayed-Multicast" framework and present an optimal resource allocation algorithm that minimizes the total bandwidth required to service a set of requests.

I. Introduction

With the widespread introduction of cable and Digital Subscriber Line (DSL), broadband to the home is now a reality. This has opened up a whole new set of online services previously unavailable to the consumers due to the bandwidth limitation in the "last-mile". One much sought after application is" video-on-demand" (VoD) where 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 on demand.

However, due to the high bit rate requirement for the display of a video, the aggregate network bandwidth required for a large number of viewers makes it difficult to provide a scalable VoD system. To address thatp roblem, we developed a new transmission technique termed "delayed-multicast" protocol (DMP) which was presented in [1][1]. The motivation behind delayed-multicast stems from the fact that processing power, disk and memory storage are relatively cheap in comparisonw ith network bandwidth. "Delayed-multicast" is similar to multicast but with one main difference being the use of a circular buffer at fan-outp oints in the network to service requests with differents tarting times. The delaying of each transmission is achieved by buffering for the required amount of time such that each client receives its data at its requested time (hence then ame "delayed-multicast").

As an example,c onsider the scenario illustrated in Fig. 1 where four requests for the same movie are made to the video server but with different starting times. Traditionally, the server will need four transmission streams to service each request as seen in Fig. 1(a). However, if the starting times are known in advance (i.e. offline cases), the bandwidth requirement from the video server to the intermediate router can be reduced to just one stream. The scalability of delayed-multicast is evident when we extend the simple topology in Fig. 1 to one where there are many more routers which can perform the buffering

[1] In that paper we used the term "scheduled-multicast" but we later discovered that the term has already been used in many other contexts. To avoid confusion, we decide to change the name to "delayed-multicast"

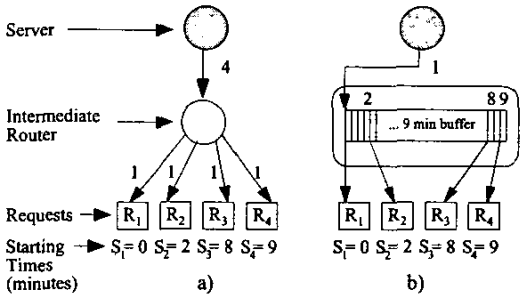

Fig.1 .a) An example scenario with four requests of the same video clip butw ith different starting time. b) An intermediate router employing a buffer to service the four requests requires only one transmission stream from the server.

In this paper we will describe a scalable framework for DMP and present an optimal offline algorithm for allocating buffer to minimize the bandwidth requirement in a three-level tree topology. The three-level tree algorithm then serves as a fundamental building block for the online algorithms.

The paper is organized asf ollows. Section 2 provides a brief survey of the related work in this area. Section 3 describest he network model for delayed-multicast. Section 4 presents the offline and online algorithms which aim to minimize the overall bandwidth requirement. Section 5 concludes the paper with several suggestions for future work.

II. Related Work

A common technique being used to minimize the aggregate bandwidth and server load is through the use of multicast. A thorough survey of the multicast techniquei s presented in [2]. However, multicastr equires thata ll users must receive the data at the same time. Thus no savings can be made if 2 or more users request the same video clip but with different request times. Batching[3], [4] tries to minimize this problem byg rouping requests and serving them together at regular intervals. This type of service is sometimes referred to as "Near Video-on-Demand" (NVoD). However, for batching to be effective, it requires a large interval leading to long waiting times for the users.

A different approach to providing NVoD for popular videos is "Pyramid Broadcasting"(PB) [5]. In this technique, videos are segmented in K segments of geometrically increasing sizes and the bandwidth is evenly divided into K logical channels, with the i^{th} channel being used to broadcast the i^{th} segment of all the videos in a sequential manner. To display a video, the client downloads and

0-7803-7632-3/02/$17.00 ©2002 IEEE

displays the first segment while at the same time downloads and buffers the next segment on disk. When it is time to display the second segment, the client retrieves it from disk and at the same time "tunes" into the third channel to receive and 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 waiting time before a client can begin watching a video. Aggarwal et al. [6], and Hua et al. [7] extend this technique to minimize disk space and I/O bandwidth requirements at the clients' ends.

However, PB and related schemes only provides NVoD, not true VoD, and are applicable only in a broadcast environment. For those reasons, a technique called "patching" was introduced by Hua et al. in [8]. In this method, later requests are serviced immediately using a "patching" multicast. The later clients subscribe to both the patching multicast and an earlier regular multicast, using the former for immediate display and buffering the later 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 can be stopped. An optimal threshold T was derived in [9] where if a request which arrives at time $t > T$ later than an earlier regular multicast then that request will be serviced from a new regular multicast rather than from a patching one. To further improve data sharing when t is greater than the client's available buffer, Sene t al. introduced a new algorithm called Periodic Buffer Reuse (PBR) [10].

A requirement of PB and patching is that the client needs to have bandwidth of at least twice the bit rate of the clip. This problem was addressed by Eager et al. in [11]. They introduced a modified form of patching termed "bandwidth skimming" that can be employed anytime the client's bandwidth is greater than clip's playback rate - the extra bandwidth is referred to as the skimmed bandwidth. A partition algorithm was presented wheree ach stream/channel is divided into k sub-channels, each carrying $1/k$ of the stream data. Assuming the client's skimmed bandwidth is sufficient for only one extra sub-channel, then during the first t period, it would receive data from all the sub-channels of the patching stream in addition to the first sub-channel from the last regular stream which is buffered on disk. During the second period $2t$,i tw ould stop receiving the first sub-channel from the patching stream since this data is now in the buffer, and use the extra bandwidth to retrieve the second sub-channel from the regular stream. This process continues until kt periods later at which time no sub-channels are needed from the patching stream and the client is considered to be "merged" with the last regulars tream. To address the issue of packet loss recovery, Mahantie t al extended the work on PB and bandwidth skimming and presented Reliable Periodic Broadcast and Reliable Bandwidth Skimming protocols in [12].

A common factor in all of the above techniques - PB, patching, and bandwidth skimming - is thatt he client musth ave available a large buffer, and the bandwidth in the "last-mile" needs to be greater (and in most cases at least twice) the clip's playback rate. By pushing the buffering into the network, our technique allows greater bandwidth savings 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.

III. Network Model

In our previous prototype presented in [1],t he tasks of delayed-multicast protocol (DMP) were carried out at the routersw hich required extra processing powera nd memory. However, because these resources are both scarce and not easily upgraded, we present a new architecture illustrated in Fig. 2 to overcome that limitation. The tasks of DMP are now shifted to dedicated machines (termed video gateways)m ade from cheap off-the-shelves PC components, and are connected to the router via a high speed link. The video gateways perform the buffering and transmission of the data while the gateway controller manages the gateways and co-ordinates the resource allocation between them. This new architecture allows DMP to be deployed in current networks and provides a scalable path for upgrade as the load increases.

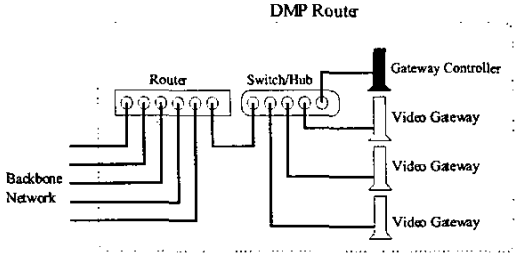

Fig. 2.D MP Router Architecture

Ideally, by employing DMP at every router in the network hierarchy, bandwidth savings can be maximized. In practice, this would not be feasible due to factors such as costs and the increased latency. A more realistic scenario is where routers employing DMP are sparsely placed at key positionsi n the path from the servers to the clients. For example, they can be placed near the video servers to reduce their I/O loads and at the head-ends to minimize the aggregate network traffic. The DMP-enable routers form an overlay network and it ist his overlay network which we use in our network model.

For the purpose of the algorithms, the network is a simple, connecteda ndu ndirectedg raph$G = (V, E)$ consisting of $n = |V|$ vertices corresponding to the DMP-routers, and $m = |E|$ edges corresponding to the links between the routers. We assume at most one edge between any pair of vertices. We do not consider non-DMP routers since one cannot perform any buffering at thesen odes, but we do make the simplification that the bandwidth available on the path between two DMP routers is the smallest bandwidth available on any link in that path.

Associated with each edge e is bandwidth B_e and likewise for each node v, C_v denotes the memory capacity available at that node.T he set of all the requests is R.F or a given request, $r \in R$, $o(r)$ specifies the originating node, $c(r)$ the clip id, and $t(r)$ the time of ther equest. For convenience, them emory capacity is measured in time units and the bandwidth is measured by the number of streams. Currently, we assume all clips have the same required bit ratef or streaming. In the following sections, we will present algorithms which utilize available buffer space with the aim of minimizing total system traffic. The optimization is subjected to two constraints: firstly, for each vertex $v \in V$ the buffer space allocated at v must be less than the available buffer space, denoted by C_v and secondly, for each edge $e \in E$ the bandwidth used over e must be less than the available bandwidth, denoted by B_e.

IV. BANDWIDTH OPTIMIZATION

The following section describes algorithms that specify how buffers are allocated in order to minimize the total bandwidth required to service a set of requests. The algorithms are separated intoo ffline - requests times are known a priori - and online algorithms. The former is analogous to situations where users can pre-book their viewing times while the later is simply the case of providing video-on-demand. While it may seem that the offline algorithm has limited applicability, it serves as a fundamental part of the online algorithm. Thus an optimal result for the offline problem is of significant importance for the online case. Note that due to space limitation, performance evaluations of the algorithms are omitted.

A. *Offline Algorithm*

The algorithm presented here is restricted to a three-level tree topology. Without loss of generality, we assume that the root has only one child which we refer toa s the chandelier topology, illustrated in Fig. 3. If this is not the case, we can solve the problem on each tree obtained by deleting all but one sub-tree of the root, and at the end by combining the solutions of these sub-problems. In the chandelier topology, only one DMP-router is placed at the intermediate node, V_2. This is analogous to only placing a DMP-enabled router at each head-end to reduce the numbero f streams required to service the requests or placing it just below the video server to minimize its I/O load.

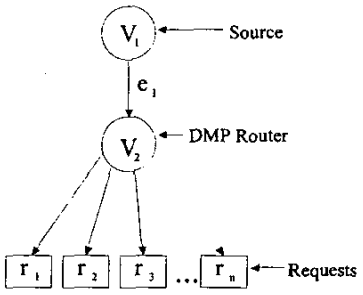

Fig. 3." Chandelier" topology: Buffering is limited to V_2.

To determine the allocation of buffers so that the minimum total bandwidth is used, we present Algorithm *Chandelier_min_traffic* which runs in polynomial time with respect to then umber of requests. The input is a set, Z, of ordered sets. Each ordered set, $z_i \in Z$, contains the request times that share the same clip, ordered in a non-decreasing sequence. For convenience, we let i denote the id of the clip. There are $k = |Z|$ different clips among all the requests.

$$Z = \{z_i : z_i = \{t(r) : c(r) = i, r \in R, 1 \le i \le k\}\} \quad (1)$$

The algorithm first determines the total amount of buffer space required if the minimum number of streams is used (i.e.o ne stream per clip).I f that buffer space is available then that is the solution.I f the buffer space is not available, then a new stream is added and the buffer space required is recalculated. The required buffers pace is kept to a minimum by searching for consecutive requestso f the same clip with the largest time difference, and removing the buffering between them. The new stream is then used to service the later request. If the new required buffer spaces till exceeds the available buffer space, another stream is added and the process repeats until either a solution is achieved or the required number of streams exceeds the available bandwidth, in which case no solution is possible.

Algorithm. ***Chandelier_min_traffic***

```
Input: Z
#Total buffer required if only one stream
#used per clip
Max_Buff ⇐ 0
for all z∈Z do
  Max_Buff ⇐ Max_Buff + (t_|z| − t_1) where t_|z|, t_1 ∈ z
end for

Space_Req ⇐ Max_Buff
#One stream is started and buffered from the
#earliest to the last request time for
#each clip
for i = 1 to |Z| do
  Start a stream at t_1 ∈ z_i for a buffer of size
  t_|z_i| − t_1
end for
if Space_Req ≤ C_v2 then
  #We do not need to allocate any more
  #streams since all the requests can be
  #serviced by the buffers
  return success
end if

#Create the set of time gaps for all clip
X ⇐ {}
for all z∈Z do
  #Create the set of time gaps between each
  # consecutive requests for a clip
  x_i ⇐ {}
  for j ⇐ 2 to |z| do
    x_i ⇐ x_i ∪ {(t_j−1, t_j)} where t_j, t_j−1 ∈ z
  end for
  X ⇐ X ∪ {x_i}
end for

#Increase 1 stream at a time to serve the
#clip with the largest gap between
#consecutive requests
for i ⇐ |Z| + 1 to B_c1 do
  y ⇐ largest gap tuple in X, where gap(a,b) =
  b − a
  Remove tuple y from the respective x∈X
  Assuming y = (t_j−1, t_j), start a stream at t_j
  Remove the buffer space allocated between
  t_j−1 and t_j
  Space_Req = Space_Req − gap(y)
  if Space_Req ≤ C_v2 then
    return success
  end if
end for
return failure
```

The idea behind Algorithm *Chandelier_min_traffic* is best illustrated by applying it to an example. Assuming we have 2 sets of request times for2 different clips: $Z = \{z_1, z_2\}$ where z_1 = {0,3,7,15,20,23} and z_2 = {2,9,11,13,22,25}. We also assume available buffer space, C_{v2} = 25 minutes, and available bandwidth is sufficient for 8 streams.

The algorithm first calculates *Max_Buff* = (23 – 0) + (25 - 2) = 46 min. The corresponding set X is{ x_1,x_2} where x_1 = {(0,3), (3,7), (7,15), (15,20), (20,23)} and x_2 = {(2,9), (9,11), (11, 13), (13,22), (22,25)}. Since the available buffer space is less than *Space_Req*, the algorithm continues by increasing the number of streams by one at a time. Table I illustrates the result of the algorithm aftere ach iteration. When the number of streams reaches 5 the algorithm terminates since the minimum space required is less than the available buffer space. We will now prove that the allocation of buffer guarantees the minimum total traffic.

Allocation of stream i	Largest Gap Tuple	Space Req.	Stream Start Time
3	(13, 22) from x2	37	{0} clip 1, {2, 22} clip 2
4	(7, 15) from x1	29	{0, 15} clip 1, {2, 22} clip 2
5	(2, 9) from x2	22	{0, 15} clip 1, {2, 9, 22} clip 2

TABLE I
ITERATION RESULTS FOR ALGORITHM *CHANDELIER_MIN_TRAFFIC*

Lemma: *For the chandelier, given i streams, the minimum buffer space required is Max_Buff - $\sum_{y \in Y} gap(y)$ where Y = {i - |Z| tuples with the largestg ap in X}.*

Proof: If $i < |Z|$ then clearly one cannot service all the requests. If $i = |Z|$ then there is only one solution which is to create a bufferf or each clip where the buffer's size stretches from thee arliest request timet o the latest request time for that clip, and a stream starting at thee arliest request time. The total buffer size is *Max_Buff*. At thisp oint we note that for any tuple, $y = (t_{j-1}, t_j)$ if a stream is started from t_j then one can save the buffer space between t_j and t_{j-1}.I f there is an extra stream available, i.e. $i = |Z| + 1$, the question is where to place the extra stream so that the maximum buffer space saving is achieved. Clearly, this cano nly be achievedb y selecting the tuple, $y \in X$, with the largest *gap*. Thus given m extra streams, $i = |Z| + m$,i tf ollows thatt he maximum buffer space saving is achieved by selecting m tuples from X whose gaps are the largest. Let this set of largest tuples be Y, the maximum buffer space saving is given by $\sum_{y \in Y} gap(y)$. The minimum buffer space required is then *Max_Buff* - $\sum_{y \in Y} gap(y)$. □

Theorem: *Algorithm Chandelier_min_traffic finds the solution which minimizes total system bandwidth required or there is insufficient bandwidth and buffer space to service all requests.*

Proof: The algorithm starts with the minimum bandwidth by allocating one stream per set of requests for a clip. If the available buffer is less than the required amount, the algorithm gradually increases the number of stream one at a time. At each iteration, the placement of the streams guarantees the minimum buffer space is allocated (from the lemma). If this minimum buffer space is less than the available buffer space, then this is a solution which uses the minimum bandwidth and the algorithm stops. The algorithm stopsw ith no solution when the number of streams required is greater than the available bandwidth, B_q . □

B. *Online Algorithms*

For the chandelier topology, we are motivated by the ideas of patching as described in [8] to handle the online cases. However, rather than buffering att he client's disks,t his is now done at the internaln ode. This has the advantages that the required last-mile bandwidth is reduced to be the same as the clip's playback rate and it lowers the costs of the set-top-boxes.

The basic idea is to service new requests immediately with a stream from the server as they arrive (we refer to these streams as "immediate" streams). An "optimization" phase, presented in Function Chandelier_optimize,i s run periodically (every W minutes) at the internal node to minimize the total bandwidth required.

Function. *Chandelier_optimize*
```
Let R_W be the set of requests arriving
during the last W minutes being serviced
by ''immediate'' streams.
Let R_fin be the set of requests finished
viewing during the last W
R ⇐ R\R_fin ⋃ R_W
Run Chandelier_min_traffic() on R
Transform the current buffer allocations
to that specified by Chandelier_min_traffic()
and remove any ''immediate'' tags for the
streams servicing these buffers.
```

The operations required to perform the transformation is illustrated in Fig. 4. Both "resize" and "de-merge" can be performed easily with immediate effect. Note thatf or the de-merge operation, at leasta n extra stream is required to service the seto f requests. The "merge" operation is different since it needs time to bridge the temporal difference before itc an complete; hence our use of the patching technique. Using the example in Fig. 4 when the merge operation is initiated, the firstb uffer is extended to cover the gap between $t(r_j)$ and $t(r_k)$ and at the same time the second buffer is extended from $t(r_j)$ to $t(r_m)$.T he streams at $t(r_k)$ and $t(r_m)$ are marked as the "patching" stream. After additional time $max(t(r_k) - t(r_j), t(r_m) - t(r_j))$ minutes,t he merge is complete and the patching streams are terminated. It is trivial to see that the operations "resize", "de-merge" and "merge" are sufficient to transform any seto f buffer allocations to a new set of buffer allocations.

To achieve multi-level buffering in a general tree, we follow a distributed method where each internal node runs its periodic optimization phase independently of each other, on the chandelier subtree consisting of its parentn ode, itself, and its immediate children.

For internal nodes just above the leaves, they use the optimization technique outlined in Function *Chandelier_optimize*. For all other internal nodes they adopt a modified optimization function presented in Function *Chandelier_internal_optimize*. The key difference between the two optimization functions is that the later use the times of the streams servicing the children as the request times for Algorithm *Chandelier_min_traffic*.

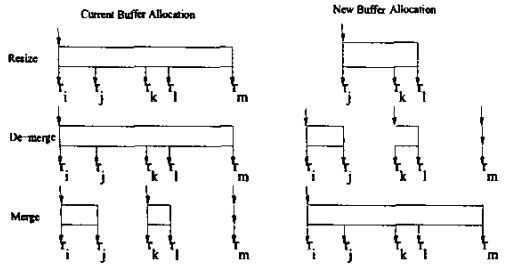

Fig.4 . Possible transformation operations. Note $t(r_i) < t(r_j) < t(r_k) < t(r_m)$ and $c(r_i) = c(r_j) = c(r_k) = c(r_l) = c(r_m)$

Furthermore, rather than using the times of all the streams passing through the current node as the request times, we exclude the streams which the immediate children nodes have marked as either being "immediate" or "patching" streams. The rationale behind the exclusion is that these streams are short-lived and it serves no purpose to include them in the optimization process.

Function. *Chandelier_internal_optimize*

```
Let R_W be the set of times for streams
whose ''immediate'' tags have been removed
by the children during the last W minutes.
Let R_fin be the set of streams which has
finished the last W
R ⇐ R\R_fin ⋃ R_W
Run Chandelier_min_traffic() on R
Transform the current buffer allocations to
that specified by Chandelier_min_traffic() and
notify the parent node the removal of any
''immediate'' tags for the streams servicing
these buffers.
```

V. Conclusion and Future Work

In this paper, we have introduced a new architecture for delivering VoD using our delayed-multicast technique. The architecture is scalable and overcomes the limitation of our previous prototype which required extra processing power and buffer capacity oft he router. This is done by shifting the work ofD MP onto dedicated machines, thus allowing DMP to be used in current networks.

Along with the new architecture, we presented and proved an optimal algorithm to minimize total bandwidth requirementf or the offline, 3-level tree problem. For the online problem, we introduced a technique similar to patching to minimize total system bandwidth requirement for VoD services. Our technique has the following advantages over traditional patching: it does not require large buffer space at the client's end, and the client's minimum bandwidth is reduced to the clip'sp layback rate. Furthermore, for patching to operate properly, a thresholdv alue, T [9], [10] is requiredt o determine when a new request should be serviced by a new stream or be "patched". This value, T , is dependent on the client's buffer size, request rate and the length of the clip. In contrast, our technique does not rely on any threshold value but uses a periodic "optimization" routine (which makes use of our optimal offline algorithm) to continually push the system towards the optimal state. More importantly, where a more general tree topology exist, we show that further bandwidth savings can be achieved by buffering at higher levels. The optimization technique for a general tree operates in a distributed manner and are therefore more applicable in practice.

Our future work focuses on developing and evaluating algorithms,a nd heuristics for minimizing resource requirements for more general network structures. In particular, we are looking at developing an appropriate cost function that will allow optimization to be done not just on the bandwidth but also on the buffer space. On the architectural side, we are investigating how to incorporate the new DiffServ and IntServ framework to provide the necessary Quality of Service guarantees for our VoD service. Finally, we will also look at how to provide some VCR functionality given the availability of the internal buffers.

References

[1] H. El-Gindy, C.N guyen, and A. Symvonis," Scheduled-Multicastw ith Application in Multimedia Networks," IEEE InternationalC onference on Networks, 2000.

[2] S. McCanne, "Scalable Multimedia Communication: Using IP Multicasta nd LightweightS essions," *IEEE Internet Computing*, vol. 3, pp. 33-45,1 999.

[3] C. Aggarwal, J.W olf, and P. S. Yu, "On Optimal Piggyback Merging Policies for Video-on-Demand Systems.," Conference on Measurementa nd Modeling of Computer Systems (SIGMETRICS), Philadelphia, 1996.

[4] A. Dan, D.Sitaram, and P. Shahabuddin," Scheduling Policies for an On Demand Video Server with Batching," ACM Multimedia Conference, San Francisco, 1994.

[5] S. Viswanthan and T. Imielinski, "Metropolitan Area Video-on-Demand Service Using Pyramid Broadcasting," *Multimedia Systems*, vol. 4, pp. 197-208, 1996.

[6] C. Aggarwal, J.W olf, and P. S. Yu, "A Permutation-Based Pyramid Broadcasting Scheme for Video-on-Demand Systems," IEEE Multimedia Computing and Systems Conference,H iroshima, Japan, 1996.

[7] K. A. Hua and S. Sheu, "Skyscraper Broadcasting:A New Broadcasting Scheme for Metropolitan Video-on-Demand System," Proc.o f ACM SIGCOMM, 1997.

[8] K. A. Hua, Y. Cai,a nd S. Sheu," Patching: A Multicast Technique for True Video-on-Demand Services," ACM Multimedia Conference, Bristol,U K,1 998.

[9] Y. Cai, K. A. Hua,a nd K.V u, "Optimizing Patching Performance," Proc. SPIE - Multimedia Computing and Networking, Santa Clara, CA, 1999.

[10] S. Sen, L. Gao, J. Rexford, and D. Towsley," Optimal Patching Schemes for EfficientM ultimedia Streaming," 9th InternationalW orkshop on Network and Operating Systems Support for DigitalA udio and Video, NJ,1 999.

[11] D. L. Eager, M. K. Vernon, and J. Zahorjan," Bandwidth Skimming:A Technique for Cost-Effective Video-on-Demand," Proceedings SPIE - Multimedia Computing and Networking, Santa Jose, CA, 2000.

[12] A. Mahanti, D. L. Eager,M . K. Vernon, and D. Sundaram-Stukel," Scalable On-Demand Media Streaming with Packet Loss Recovery," Proceedings of ACM SIGCOMM, New York, NY, 2001.