Multi-level Caching With Delayed-Multicast for Video-on-Demand

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

Antonios Symvonis
Department of Mathematics
University of Ioannina, Greece.
symvonis@cc.uoi.gr.

ABSTRACT
Delayed-Multicast is a novel transmission technique to support Video-on-Demand. It introduces buffers within the network to bridge the temporal delays between similar requests thus minimizing the aggregate bandwidth and server load. This paper introduces an improved online algorithm for resource allocation with Delayed-Multicast by utilizing prior knowledge of each clip's popularity. The algorithm is intended to be simple so as to allow for deployment at multiple levels in a distribution network. The result is greater backbone traffic savings and a corresponding reduction in the server load.

KEY WORDS
Video-on-Demand, Multimedia Networking

1 Introduction

As the level of broadband penetration increases, many applications which were limited by the "last-mile" bottleneck are now becoming more accessible to the users. A large proportion of these applications such as corporate webcasting, and video-on-demand (VoD) for home entertainment or distant learning, requires the ability to stream high volume of continuous media. Currently, a common technique is to provide a dedicate unicast stream from the server to the client. However, this solution pushes the bandwidth bottleneck back into the network backbone and increases the demands on the server. The result is a non-scalable system both in terms of network bandwidth requirements and server loads.

Many techniques have been proposed to overcome the problem mentioned above. A common factor in many of these techniques is the exploit of different clips' popularity in minimizing the resource requirement. Batching [1, 2] is one such technique which groups requests into "batches" and serves each batch through the use of a single multicast stream. However, for effective resource savings, batching requires a large time interval - to maximize the number of users within each group - leading to potentially long wait times for users.

To reduce the maximum wait-time to a small fraction of the clip's length, Viswanathan and Imielinski [3] introduced "Pyramid Broadcasting" (PB) scheme. With PB, videos are segmented in K segments of geometrically increasing size and the broadcast 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. PB also requires that the time to download any i^{th} segment must be less than the time to display the $(i-1)^{th}$ 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 "tune" into 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. However, the drawbacks is that the disk bandwidth and the buffering space at the client's end can be very large.

To avoid the wait-time completely Hua et al. [4] introduces a technique termed "patching". It also utilizes disk space at the client's set-top box (STB) to minimize network bandwidth. With patching, initial requests are serviced using a regular multicast. Subsequent requests for the same clip are serviced immediately with a stream from the server known as a "patching" stream. However, rather than subscribing only to its patching channel, the later client also subscribes to the nearest earlier regular multicast, using the patching stream for immediate display while 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 stream is terminated. Variations to the patching scheme was introduced in [5] to reduce the required last mile bandwidth requirement to below twice the bit rate of the video clip, and in [6] to address the issue of packet loss recovery.

Recent works [7, 8, 9] introduce the concept of proxy caching. A straight forward application of proxy caching is to permanently cache portions of popular clips. The advantage of this is that it can reduce latency, and mask network jitters, in addition to reducing resource requirement. More advanced forms of proxy caching make use of batching and patching at the proxy to further reduce network bandwidth requirement.

In this paper, we present our protocol, termed "Delayed-Multicast", which performs buffering of streams within the distribution network, thus reducing the need for large client-side buffers. Furthermore, the "last-mile" bandwidth only needs to be as large as the clip's play-

back rate. Accompanying the protocol is an online (request times are not known *a priori*) distributed algorithm which minimizes the aggregate backbone bandwidth requirement and server load.

The outline for the rest of the paper is as follow: Section 2 describes the operations of "Delayed-Multicast" and our system architecture. Section 3 introduces the online distributed algorithm. Section 4 discusses experimental results. Finally, we conclude in Section 5.

2 Delayed-Multicast Protocol

The "Delayed-Multicast" Protocol (DMP) employs circular buffers within the networks to allow the partial caching of data streams. Later requests for the same data can be serviced by the buffer rather than starting a new stream from the server. Consequently, there is no need to have a large buffer space at the client, and the "last-mile" bandwidth does not need to be greater than the clip's play back rate.

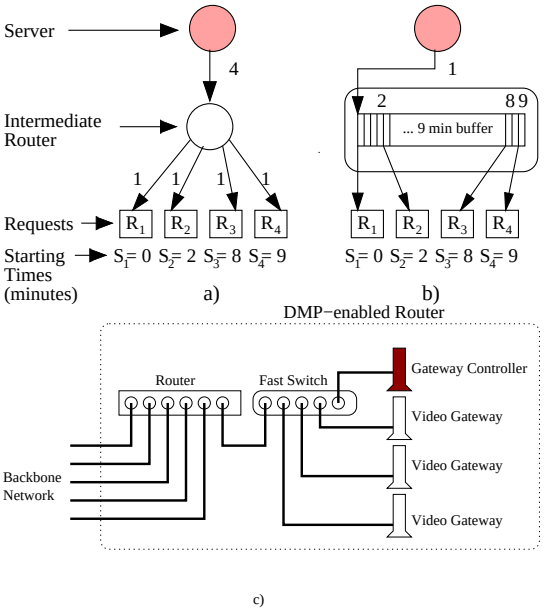

Figure 1. a) Four requests requiring 4 streams from the server. b) Employing Delayed-Multicast at the intermediate router can reduce the required bandwidth from the server to just 1 stream. c) A DMP-enabled router.

An illustration of Delayed-Multicast can be seen in Figure 1 a) and b). Traditionally, servicing the four requests requires four streams from the video server. If there is a buffer in the intermediate node in the transmission path sufficient to bridge the temporal differences, the required bandwidth can be reduced to one stream.

DMP does not require special router hardware since the work of buffering and scheduled transmission is handled by dedicated machines (video gateways) placed physically close to the router. The whole set-up, shown in Figure 1 c), is referred to as a DMP-enabled router.

While advances in disk technology allow abundant storage space at a very cheap price, utilizing disks as the source of buffer space is still not feasible. The main obstacle is the poor seek time of disks relative to memory which severely restricts the number of concurrent streams. Instead, our architecture relies on memory to perform the buffering. In terms of costs, it is now quite feasible to build a system with up to 1GB of memory for under $1000. Multiple video gateways can be easily added in future as the demand increases to provide more buffer capacity.

An aim of DMP is to perform buffering at multiple levels in the distribution tree since this would lead to greater savings in terms of aggregate backbone traffic and server load. As an example, one can place a DMP-enabled router at each head-end in hybrid fibre cable (HFC) networks, or DSLAM Multiplexer for Digital Subscriber Line (DSL) networks, as well as another DMP-enabled router next to the video server. DMP then forms an overlay tree network consisting of the DMP-enabled routers, shown in Figure 2.

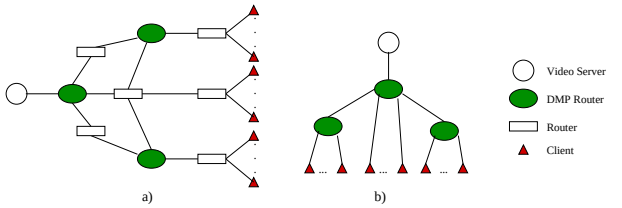

Figure 2. a) Placement positions for DMP-enabled routers in a distribution network b) Resulting overlay tree network.

Techniques for building efficient overlay network is outside the scope of this paper. For the moment we simply assume each DMP-enabled router in the distribution tree has a fixed knowledge about the location of its parent DMP-enabled router. This is quite valid since DMP is to run on a distribution network whose topology is not expected to change greatly.

The distributed operations of DMP can be described as follows. At the first internal node when a request arrives, an algorithm is run to determine whether or not to service the request from an existing buffer, or by requesting a new stream from the parent. In the latter case, the node creates a buffer whose size is determined by the employed algorithm, and forwards a request for the same clip to its parent. When the parent node starts sending the requested stream, the buffering node places the incoming data into the appropriate buffer. At the same time, for all the clients being serviced from the same buffer, it transmits data from that buffer but at the appropriate offset for each client.

2.1 Resource Allocation

An important question which arises is how best to allocate the limited buffer space to minimize the number of streams required from the server above. We start with a simple

three-level overlay tree, as shown in Figure 1. In this case, there is an optimal offline (request times are known *a priori*) algorithm termed "Chandelier Algorithm". We assume that all clips have the same bit rate. Let B denotes the maximum bandwidth available from the server to the intermediate node measured in number of streams, and S the available buffer space at that node measured in number of minutes. For each request, $r \epsilon R$, $t(r)$ and $c(r)$ are the time and clip id respectively of that request. There is a total of N different videos. The input to algorithm Chandelier is a set, Z, of ordered sets. Each ordered sets, $z_i \epsilon Z$, contains the request times for the same clip, ordered in non-decreasing sequence.

Algorithm Chandelier

```
Input: Z where Z = {z_i : z_i = {t(r) : c(r) = i, r ϵ R}, 1 ≤ i ≤ N}
#Total buffer required if only one stream
#is used per clip
Space_Req ⇐ 0
for all zϵZ do
  Space_Req ⇐ Space_Req + (t_|z| − t_1) where t_|z|, t_1 ϵ z
end for
Max_Buff ⇐ Space_Req
for i = 1 to |Z| do
  Start a stream at t_1 ϵ z_i for a buffer of
  size t_|z_i| − t_1
end for
if Space_Req ≤ S then
  return success
end if

X ⇐ {}
for all zϵZ 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} ϵ z
  end for
  X ⇐ X ∪ {x_i}
end for

for i ⇐ |Z| + 1 to B do
  y ⇐ largest gap tuple in X, where
  gap(a,b) = b − a
  Remove tuple y from the respective xϵX
  Assuming y = (t_{j−1}, t_j), start a stream at
  t_j
  Deallocate buffer space allocated be-
  tween t_{j−1} and t_j
  Space_Req = Space_Req − gap(y)
  if Space_Req ≤ S then
    return success
  end if
end for
return failure
```

Intuitively, the algorithm minimizes the required upstream bandwidth by serving all requests for the same clip from one buffer which requires only one stream. If there is not enough buffer space, an extra stream is required and that stream is chosen to start at the point where there is the largest gap between consecutive requests for the same clip, chosen amongst all the clips. This frees up the maximum possible buffer space and the algorithm continues until the required buffer space can be satisfied by the available buffer space.

Lemma 1 *For the chandelier, given i streams, the minimum buffer space required is $Max\_Buff - \sum_{y \epsilon Y} gap(y)$ where $Y = \{i - |Z|$ tuple with the largest gap in $X\}$.*

Theorem 1 *Algorithm* Chandelier *finds the solution which minimizes total upstream bandwidth required and that solution uses the least amount of buffer space, or there is insufficient bandwidth and buffer space to service all requests.*

Proofs for the above theorem and the associated lemma can be found in [10]. While the Chandelier algorithm has the nice property that it is optimal, it is quite limited. One problem is trying to generalize the algorithm to a distribution tree with multiple buffering levels, but finding an optimal solution here is NP-complete [10]. Another problem is that the Chandelier algorithm is a centralized one requiring global knowledge of resource availability. This is not applicable in a distributed environment. However, the main drawback of the Chandelier algorithm is the requirement of prior knowledge of request times, limiting its applicability to only situations where users must place their requests a long time in advance.

In the next section we present an online algorithm which does not have the drawbacks mentioned above. Our aim is to develop a simple, distributed algorithm which can be applied at multiple levels within the distribution tree and whose performance approaches the optimal offline Chandelier algorithm.

3 Online Algorithm

We first examine the algorithm for the online three-level topology since that forms the basis of the distributed algorithm. The idea behind the handling of online requests is as follows. When the node receives the first request r_i for a particular clip, it creates a buffer of size s_i and forwards that request to the server. For subsequent requests r_{i+1} of the same clip, if $t(r_{i+1}) \leq t(r_i) + s_i$ then clearly, it can be serviced from that buffer otherwise it needs a new buffer with a corresponding stream from the server. The question is what should s_i be to maximize the number of requests that can be served from it given the limit of available buffer space S at that node.

In an online scenario, while the exact request times are not known *a priori*, one can assume the arrival times of video requests to follow a Poisson distribution with an average number of requests λ over a period P. Furthermore,

in the case of a video library rental, certain videos such as new releases are more popular than others. If the videos are ordered according to their popularity, the probability, p_i, that a request selects a particular video, i, $i = 1, 2, ..., N$, follows the Zipf distribution given by:

$$p_i = 1/i^z K_N \text{ where } K_N = \sum_{i=1}^{N} 1/i^z \quad (1)$$

The skew factor, z, that has been found to closely match the selection probability at a video library rental is 0.271[1].

It can be seen that as more buffer space is allocated to a video, the aggregate bandwidth savings for that video also increases because there is a greater likelihood that a request will be serviced from the buffer space rather than by a new stream from the server. Thus if more buffer space is allocated to more frequent videos then more requests can be served from the buffer space, resulting in greater bandwidth savings. Similarly, allocating more buffer space to longer videos also yield more saving.

Based on these observations, for each clip i with its respective length L_i, we partition the total space available S into $S_i, S_j, ..., S_N$ where S_i is the buffer space reserved for video i and is given by $S_i = p_i L_i S / \sum_{j=1}^{N} p_j L_j$. The partition of the buffer space has the advantage that infrequent requests do not take up excessive buffer space and starving more frequent requests. Compare this with the Chandelier algorithm, we see that no differentiation is made there in terms of a clip's popularity. As a result, infrequent requests can lead to poor solutions since they will exhaust all the available buffer space early on.

However, if we apply the partition technique, and run the Chandelier algorithm separately for each clip, i, with the available buffer space set to the corresponding S_i, the resultant buffer allocation is the optimal for that clip albeit only for the offline cases. In other words, this is equivalent to running the Chandelier algorithm on each group of requests where the input is $|Z| = 1$ and $S = S_i$ for each clip $i, 1 \leq i \leq N$. We refer to this algorithm as the Partitioned Chandelier algorithm.

Notations used in Online DMP Algorithm	
$S_i^{allocated}$	Total buffer space allocated to sessions buffering clip i
r_{last}	The previous request for clip i
r_{start}	The request at the start of the buffer serving r_{last}.
$size(r_k)$	Size of buffer with request r_k at the start.

The aim of "Online DMP" algorithm is to try and achieve the same result as the Partitioned Chandelier Algorithm over the same time period P. The algorithm is run every time a new request arrives, at the granularity of 1 minute. The basic idea of the algorithm can be explained as follows. When the first request arrives for a video, i, it creates a new session whose buffer size is the whole buffer space allocated to video i, namely S_i. For subsequent requests, it simply checks whether or not to serve the request from the buffer or create a new session. The decision is based on a threshold T. If the inter-arrival time between the new request, r, and the last request for the same clip, r_{last}, is less than the threshold, T, the request can be serviced from the existing buffer if one exist. Otherwise the buffer of the previous request is re-sized to stop at the last request, r_{last}, while the new request is assigned to a new session whose buffer size is equal to all the free buffer space available to clip i at that point in time. This operation is illustrated in Figure 3. It is important to note that our scheme is unlike other techniques which fix the sizes of the buffers at the time they are created. Instead, we try to allocate as large a buffer as possible and then later "trim" the buffer from the back, depending on the inter-arrival threshold T. In a later section we will analyze how to determine the threshold value T.

Online DMP Algorithm

Input: request r **where** $c(r) = i$
begin
if $r_{last} = r_{start} = NULL$ **then**
  Create a new session with buffer size $t(r) + S_i - S_i^{allocated}$ **for request** r.
  $S_i^{allocated} = S_i^{allocated} + size(r)$.
  return
end if
$gap = t(r) - t(r_{last})$.
if $gap \geq T$ **then**
  Perform $resize$ **from the back of buffer of** r_{start} **where** $size(r_{start}) = t(r_{last}) - t(r_{start})$.
  Let d **= space freed from resizing buffer of** r_{start}
  $S_i^{allocated} = S_i^{allocated} - d$.
  Create a new session for r **with a buffer starting from** $t(r)$ **to** $t(r) + S_i - S_i^{allocated}$.
  $S_i^{allocated} = S_i^{allocated} + size(r)$.
else
  if $t(r_{start}) + size(r_{start}) > t(r)$ **then**
    r **can be serviced by** r_{start} **buffer so append it to that session.**
  else
    Create a new session with buffer size $t(r) + S_i - S_i^{allocated}$ **for** r.
    $S_i^{allocated} = S_i^{allocated} + size(r)$.
  end if
end if
end

For requests which have finished, the space allocated to the buffers servicing those requests are freed, and $S_i^{allocated}$ is decremented accordingly at those times. This is run in a separate process and is not shown in the algorithm.

The rationale behind using a threshold based on the inter-arrival time, rather than a fixed buffer size allocated at the start of the buffer creation, is because we want to approximate the optimal offline Partitioned Chandelier Algorithm. To see this, consider a period $P \leq L_i$, and a given set of request R of the same clip i which arrived during this time. If we put it through the Partitioned Chandelier Algo-

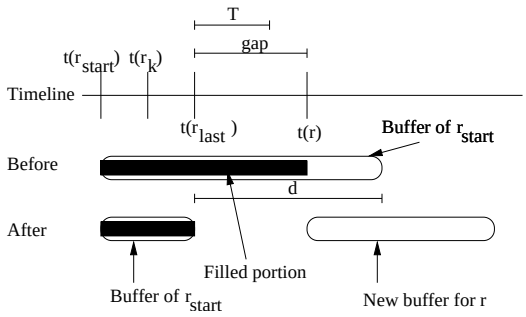

Figure 3. The results when a new request r arrives at $t(r)$ whose inter-arrival time $gap > T$.

rithm with the available buffer space set to the corresponding S_i, we get back a series of buffer allocations ordered in increasing request times, which guarantees the minimum upstream bandwidth required. Let K be the smallest "gap" (i.e. the time difference from the start of a buffer to the end of the previous buffer) amongst all the buffers. By setting the threshold $T = K$, we get the same buffer allocation when running the Online DMP algorithm on this set of requests.

3.1 Determining Threshold Value, T

The performance of the Online DMP algorithm is based on correctly choosing the threshold value T_i for each clip i. Too large a value will cause initial requests to greedily consume all the available buffer space, preventing subsequent ones to form buffers even though they may be closer together. On the other hand, if T is too small with respect to the average inter-arrival time, requests will also not be able to form buffers. In the following section, we present an approximation algorithm for the threshold T_i. We assume request arrival times follow a Poisson process with λ_i average rate of arrivals per minute.

A boundary case is when $S_i << 1/\lambda_i$, Online DMP algorithm will have no effect here because there is insufficient buffer space to bridge the temporal differences between rare requests. In this situation, it is better not to perform DMP on clip i and instead reserve space for more popular clips. This is done using the "eighty-twenty" rule, i.e. we only perform DMP on the most popular 20% of the clips.

To approximate T_i, we consider a period equivalent to the length of the movie L_i. The expected number of arrivals during this period is $\lambda_i L_i$. Assuming all the buffer space S_i is to be used during this period, then the average buffer space allocated per request is given by $S_i/\lambda_i L_i$.

Let n_i be the number of actual requests for clip i which has arrived up to this point, t, $0 < t \leq L_i$. We denote S_i^{fixed} to be the sum of all the buffer space allocated to clip i *excluding* the buffer space from $t(r_{last})$ to the end of that buffer. To ensure that T_i is not set to large so as to "hog" the buffer space, we use the condition below to determine whether or not the current request r should be part the existing buffer or start a new one:

if $S_i/\lambda_i L_i \leq (S_i^{fixed} + gap)/n$ **then**
  Create a new buffer
else
  Continue as part of the existing buffer
end if

Or to put it another way, the threshold T_i is given by

$$T_i = min \left\{ \begin{array}{l} (S_i n/\lambda_i L_i) - S_i^{fixed} \\ t(r_{start}) + size(r_{start}) - t(r_{last}) \} \end{array} \right. \quad (2)$$

Thus the threshold value T_i is not fixed but varies at each processing of new requests.

3.2 Applying Online DMP Algorithm on General Tree Network

The job of each node in the overlay tree is to receive incoming requests from registered children nodes immediately below it. To service them, each node runs the Online DMP algorithm based on the requests it receives from the children nodes. If in order to satisfy the requests, new sessions are needed, it will forward a request for the required clip to its parent. The same process occurs again in the parent node. Thus, an initial request for a clip will be forwarded hop by hop in the overlay tree with a new session created at each point.

It is important to note that for the Online DMP algorithm, it does not require complex signalling between the parent and the child node. All that is needed is the initial request from the child to the parent node. Furthermore, the only changes that can occur to the buffer is done through the *resize* operation which occurs from the back of the buffer. As a result, the request times of the streams coming from the parent node to feed into the front of that buffer does not need to change (assuming users do not quit midway through the video). The local nature of changes makes it ideal for employing Online DMP algorithm at multiple levels in the distribution tree. If they were allowed to change frequently, any optimization done at higher levels would need to be updated frequently as well. The "local" nature of any changes is ideal for employing Online DMP algorithm at multiple levels in the distribution tree.

4 Experimental Results

In this section we evaluate Online DMP through simulations. We assume that there is no defection - users do not quit midway through viewing the video clips. We also assume that there is no blocking, i.e. all requests can be satisfied. The performance metric we are interested in is the average transmission cost from the server to the DMP-enabled per client, measured in terms of clip minutes transmitted, normalized over the clip's length L_i denoted as C.

We compare "Online DMP" algorithm against three base cases: no caching or batching, batching (at 1 minute interval), and batching with traditional caching. Note that our algorithm can be described as batching with "Online DMP" since the algorithm is run at the time granularity of 1 minute. We compare these schemes in a three level topology, since that is the basis of the distributed algorithm.

In the case of no batching or caching, the average transmission cost C from the parent node over a period L_i is simply $\lambda_i L_i^2$. In the second case, we have:

$$C = \begin{cases} \lambda_i L_i^2 & \text{if } \lambda < 1 \\ L_i^2 & \text{if } \lambda \geq 1 \end{cases} \quad (3)$$

The result arises from the fact that when $\lambda \geq 1$, batching takes effect and so there is an upper limit on the number of streams required from the server. For batching with caching of size S_i (i.e. a prefix of the clip of size S_i is always kept at the intermediate node) the cost over the duration L_i minutes is

$$C = \begin{cases} L_i & \text{if } L_i - S_i = 0 \\ \lambda_i L_i (L_i - S_i) + S_i & \text{if } \lambda < 1 \text{ and } L_i - S_i > 0 \\ L_i (L_i - S_i) + S_i & \text{if } \lambda \geq 1 \text{ and } L_i - S_i > 0 \end{cases} \quad (4)$$

since for every stream coming from the server, only $L_i - S_i$ portion needs to be streamed plus the initial cost of sending the cache portion, S_i. The comparisons of these schemes with Online DMP are presented in Figure 4. The threshold values, T_i, used were dynamically determined using the approximation technique mentioned earlier.

The results for Batching & Caching are shown as straight linear line because the same requests set, generated at the start of the simulation, is reused for each S_i value. As a result, there is a linear relationship as more buffer space becomes available.

Figure 4 shows that for all λ_i and S_i, the savings achieved by Online DMP algorithm is greater than the traditional caching with batching. The only exception is when $S_i \approx L_i$, where it is obvious that the better solution is the caching of the entire clip.

Another point of interest is the performance of our buffer space partitioning scheme. As mentioned earlier we adopt the "eighty-twenty" rule do decide which clips to perform DMP on. In addition we then partition based on the popularity of the clip (i.e. using the Zipf distribution). In the simulation, we have a library of 50 different clips of different lengths, uniformly distributed between 100 min and 120 min. The Zipf skew factor of 0.271 is used.

Figure 5 shows the results of our partitioning scheme against the naive equal partition scheme (where the available buffer space is equally shared amongst all the clips), and one which simply uses the "eighty-twenty" rule (i.e. the buffer space is equally shared amongst the 20% most popular clips). The effects of the "eighty-twenty" rule clearly has the most effect with a large amount of savings achieved. Our scheme with further partitions the space for

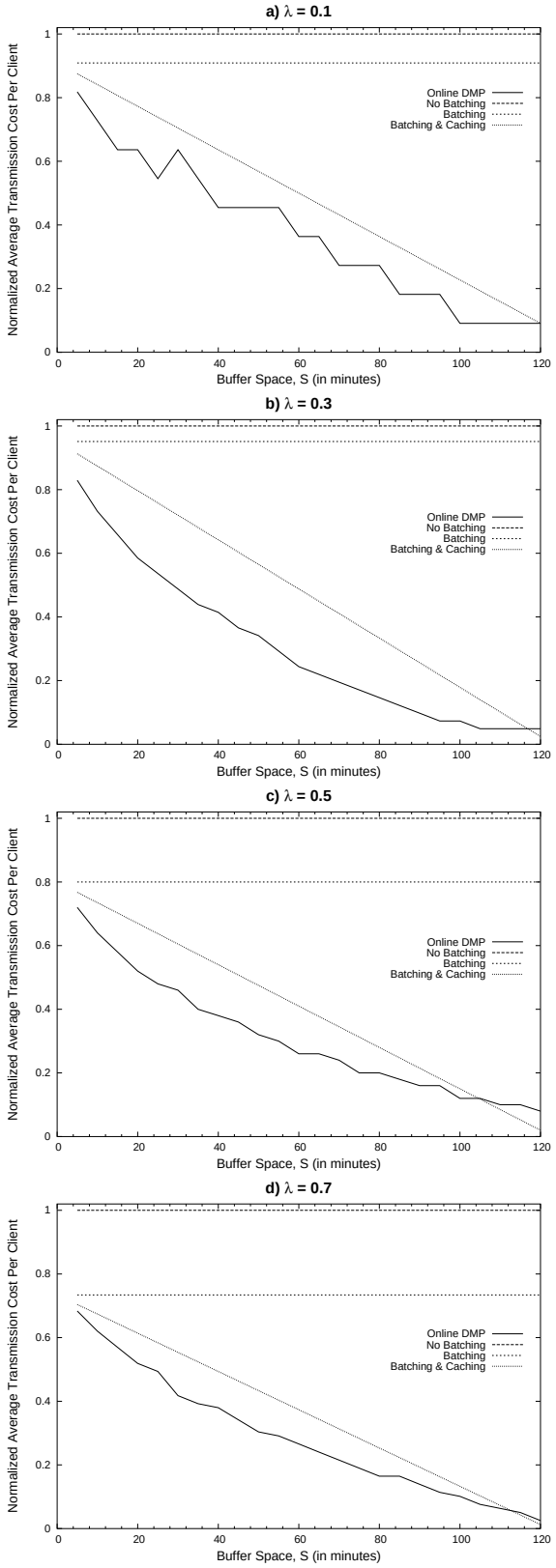

Figure 4. Comparisons of required bandwidth savings for "Online DMP" versus other transmission schemes with different arrival rates, λ_i, and available buffer space, S_i.

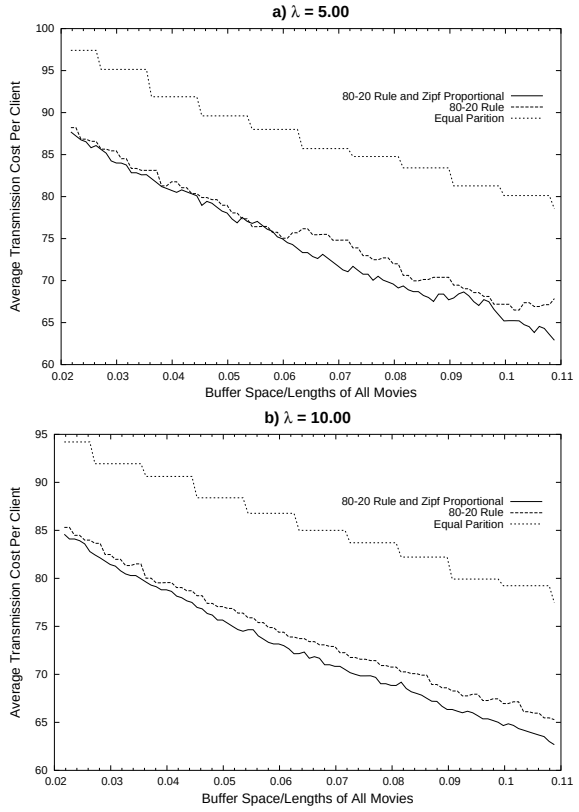

Figure 5. Comparison of different buffer allocation schemes.

each popular clip based on its popularity ranking achieves slightly more bandwidth savings.

5 Conclusion

In this paper we present an overview of the Delayed-Multicast Protocol. This is a novel protocol which relies on an overlay tree network for distributing Video-on-Demand. Its aim is to reduce the aggregate backbone traffic by buffering streams within the network. We present an optimal offline algorithm, Chandelier Algorithm, for simple three-level tree topology, which given a set of request times, allocates buffer space so that the total transmission cost from the server to the intermediate node is minimal.

To convert the Chandelier Algorithm to an online solution, we introduce a new online algorithm, "Online DMP". Our algorithm relies on a threshold value T for the inter-arrival time to determine whether or not to start a new stream or service the new request from an existing prefix cache. The simplicity of the algorithm enables it to be deployed at multiple levels in the distribution tree, thus further improving bandwidth utilization. To further improve bandwidth savings, we utilize a buffer space partitioning scheme that exploits different clip's popularity to optimize buffer allocation which in turns leads to better buffer utilization.

As on going work, we are looking at providing a DMP implementation that is compliant with RTP. Many practical issues such as start-up latency, QoS provisioning and packet loss recovery which have not been dealt with in this paper will be the focus of our future work.

References

[1] Charu Aggarwal, Joel Wolf, and Philip S. Yu, "On Optimal Piggyback Merging Policies for Video-On-Demand Systems," in *Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '96)*, Philadelphia, PA, 1996, pp. 200–209.

[2] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with batching," in *ACM Multimedia Conference*, San Francisco, CA, 1994, pp. 15–23.

[3] S. Viswanathan and T. Imielinski, "Metropolitan Area Video-on-Demand Service Using Pyramid Broadcasting," *Multimedia Systems*, vol. 4, no. 4, pp. 197–208, 1996.

[4] Kien A. Hua, Ying Cai, and Simon Sheu, "Patching: A multicast technique for true video-on-demand services," in *ACM Multimedia Conference*, Bristol, UK, 1998, pp. 191–200.

[5] Derek Eager, Mary Vernon, and John Zahorjan, "Bandwidth Skimming: A Technique for Cost-Effective Video-on-Demand," in *Proc. SPIE – Multimedia Computing and Networking*, Santa Jose, CA, January 2000, pp. 206–215.

[6] Anirban Mahanti, Derek L. Eager, Mary K. Vernon, and David Sundaram-Stukel, "Scalable On-Demand Media Streaming with Packet Loss Recovery," in *Proceedings of ACM SIGCOMM'01*, New York, NY, 2001, pp. 97–108.

[7] S. H. Gary Chan and Fouad Tobagi, "Distributed Servers Architecture for Networked Video Services," *IEEE/ACM Transaction on Networking*, vol. 9, no. 2, 2001.

[8] B. Wang, S. Sen, M. Adler, and D. Towsley, "Optimal proxy cache allocation for efficient streaming media distribution," in *IEEE INFOCOM*, New York, NY, 2002.

[9] Subhabrata Sen, Jennifer Rexford, and Don Towsley, "Proxy prefix caching for multimedia streams," in *IEEE INFOCOM*, New York, NY, 1999.

[10] Nikolaos Glinos, Doan B. Hoang, Chi Nguyen, and Antonios Symvonis, "Algorithmic support for video-on-demand based on the delayed-multicast protocol," in *9th International Colloquium on Structural Information and Communication Complexity*, Andros, Greece, 2002.