

Multicast with Cache (Mcache): An Adaptive Zero-Delay Video-on-Demand Service

Sridhar Ramesh	Injong Rhee	Katherine Guo
ST Microelectronics	Department of Computer Science	Bell Laboratories
1060 E. Brokaw Road	North Carolina State University	Lucent Technologies
San Jose, CA 95131	Raleigh, NC 27695-7534	Holmdel, NJ 07733
<code>sridhar.ramesh@st.com</code>	<code>rhee@csc.ncsu.edu</code>	<code>kguo@lucent.com</code>

Abstract

A closed-loop (demand-driven) approach towards video-on-demand services, called multicast with caching (Mcache) is discussed in this paper. Servers use multicast to reduce their bandwidth usage by allowing multiple requests to be served with a single data stream. However, this requires clients to delay receiving the movie until the multicast starts. Using regional cache servers deployed over many strategic locations, Mcache can remove the initial playout delays of clients in multicast-based video streaming. While requests are batched together for a multicast, clients can receive the prefix of a requested movie clip from caches located in their own regions. The multicast containing the later portion of the movie can wait until the prefix is played out. While this use of regional caches has been proposed previously, the novelty of our scheme lies in that the requests coming after the multicast starts can still be batched together to be served by multicast patches without any playout delays. The use of patches was proposed before, but they are used either with unicast or with playout delays. Mcache effectively hires the idea of a multicast patch with caches to provide a truly adaptive video-on-demand service whose bandwidth usage is up to par with the best known open-loop schemes under high request rates while using only minimal bandwidth under low request rates. In addition, efficient use of multicast and caches removes the need for a priori knowledge of client disk storage requirements which some of the existing schemes assume. This makes Mcache ideal for the current heterogeneous Internet environments where those parameters are hard to predict. We further propose the Segmented Mcache (SMcache) scheme which is a generalized and improved version of Mcache where the clip is partitioned into several segments in order to preserve the advantages of the original Mcache scheme with nearly the same server bandwidth requirement as the open loop schemes under high request rates.

1 Introduction

Video-on-Demand (VoD) is gaining popularity in recent years with the proliferation of high bandwidth networks. Applications using VoD include news distribution (for example, headline news from `cnn.com`), distance learning and entertainment video distribution. Deployment of new technologies in last mile access networks such as DSL and cable modem have made VoD over the Internet possible.

A typical VoD system consists of a set of centralized video servers and geographically distributed clients connected through high-speed networks. A large number of video files are stored

on the servers and played by the clients. Before accepting a client's request, the server has to reserve sufficient processing capacity and network I/O bandwidth for the video stream in order to guarantee continuous playback of the video. We use *channel* to refer to the unit of server network bandwidth needed to support one video stream. Because of the high bandwidth requirement of video streams, server network bandwidth is considered the most expensive resource in a VoD system [21]. Therefore a critical part of a VoD system design is to minimize server bandwidth requirement.

Internet traffic is bursty; recent study [10, 16] indicates that arrival rates of requests to the Web are highly variable. There are typically long periods of idle times with relatively little request traffic, mixed with short periods of high request burst. Although we do not yet have evidence that VoD requests will follow the characteristics of the Web, we believe that such characteristics are quite likely. Some preliminary study [6] suggests that the request patterns for news-on-demand services follow those of the Web (e.g., Zipf's law on web page popularity). In this paper, we examine the performance of VoD request scheduling protocols under an environment where request rates may be highly variable. In this environment, VoD service has to be highly adaptive, optimizing its server bandwidth usage to the level of request traffic.

Existing VoD schemes fall in two categories: *closed-loop schemes* [12, 11, 7, 19, 2, 8, 9, 20] and *open-loop schemes* [3, 31, 1, 22, 4]. In most closed-loop schemes, the server allocates channels and schedules transmission of video streams based on client requests using *batching*, or *patching* techniques. In batching, requests for the same video clip are delayed for a certain amount of time to serve as many requests as possible with one multicast channel [2, 8, 9, 30]. In patching, when a client issues a request for a video clip, it immediately joins an existing multicast channel A of the clip. Since it has missed the beginning part, the server establishes a new unicast channel B to send the missing part as a patch [21, 19, 28]. Thus, unicast patching does not introduce any playout delay at the client (other than network delays). However, existing patching protocols use high bandwidth under high request bursts. *Controlled Multicast* [19], the best patching protocol known to date, requires $O(\sqrt{\lambda L})$ server channels where λ is the mean arrival rate of requests and L is the mean size of video clips. For very low arrival rates, the expected number of server channels required is λL .

A desirable scheduling protocol should make the server bandwidth usage independent of the request arrival rate when the rate is high while making the bandwidth usage low when the rate is low. Open-loop schemes require constant bandwidth regardless of request rates. Recent years have seen a number of innovative open-loop schemes [31, 1, 22, 23, 17, 4] that differ on multicast schedules and server bandwidth requirements. These schemes require only $O(\log L)$ server channels, the theoretical lower bound for required server channels for VoD. However, open-loop schemes are not adaptive because the server broadcasts a constant amount of video streams regardless of whether there is an outstanding request or not. Thus, while an open-loop approach can support an unlimited number of requests for popular video clips with a constant amount of bandwidth, it wastes bandwidth when the request frequency is low.

In this paper, we propose a novel adaptive closed-loop scheme, called *Multicast Cache (Mcache)*, that uses part of the client storage space or some caches to store some parts of popular video clips

to offer zero-delay for playout and lower server and cache bandwidth requirements than the best known closed-loop schemes such as [12, 11]. Its properties are:

- Under very low arrival rates, the mean channel usage of the server tends to λL . Observe that this is the lower bound, because each request necessitates a complete playout of the video clip by the server. Under high arrival rates, the mean channels usage of the server is bounded by $O(\log L)$, independent of λ .
- Clients do not experience *any playout delay* other than network delays in receiving requested video clips. The bandwidth consumption at a client is also constant, independent of the length of video clips (each client requires at most two channels at any time).
- The amount of disk space at clients and at caches has much less impact on its performance than on that of other existing cache-based closed-loop approaches. Its tradeoffs between the resource usage (disk space and bandwidth) of the server and that of caches are much better than those of other existing cache-based closed-loop approaches.
- It does not require any a priori knowledge of client disk space.

The central idea of Mcache is to use batching, patching, and prefix caching [29] techniques with multicast to complement one another. In Mcache, a client sends its request to both the server and its local cache. The request is immediately served by the cache for the prefix of the requested clip. In the meantime, the server can batch the request with other requests arriving within the duration of the prefix. Those requests batched together are served with one multicast channel after that duration. Furthermore, clients requesting a clip after the multicast session for the clip has begun can still join that multicast session. They get the missing part from a separate stream as a patch. Since the clients receive the prefix from their caches, this patch can also be delayed for the length of that prefix. This enables the server to batch other requests for the same patch. The batched patch requests are served in another multicast session after that delay. Note that this scheme does not introduce any delay (other than network delays) from the client's point of view. Batching the requests for patch is a unique feature of our scheme.

Our goal is to reduce bandwidth usage by sharing multicast channels for the video clip and its patches among as many clients as possible. A major constraint is to offer clients instantaneous playback. Using prefix caches essentially allows the server to delay the starting time of both types of multicast without actually delaying the client's playout time, thus providing more opportunities to serve later requests with existing channels.

In this paper, we analytically estimate the server bandwidth usage of VoD services, and validate our results by simulation. Our analysis shows that the performance of Mcache in terms of the time average of the number of server channels used is usually the best among all the schemes we tested over a variety of client request rates.

This paper is organized as follows. Section 2 contains an overview of existing work on protocols for VoD systems. Section 3 provides an outline of the VoD system environment. Section 4 presents the basic Mcache multicast algorithm which uses prefix caching. Section 5 discusses a variation

of this scheme involving partitioning of video clips into segments. We provide an upper bound analysis of the segmented scheme called SMcache, and discuss a further generalization called partitioned SMcache. In Section 7, we compare the performance of SMcache with some well-known open-loop and closed-loop schemes using numerical examples. Section 8 concludes the paper.

2 Related Work

Patching was first proposed by [21] and further extended by [19, 28, 5] by optimizing the server bandwidth requirement. In [19], a threshold policy is proposed, and the resulting scheme is referred to as Controlled Multicast. Controlled Multicast is a closed-loop approach that has been shown to provide good performance under low request rates. However, for *hot* videos, the performance deteriorates, especially when the length of the clip becomes large. It is shown that the server bandwidth grows as $O(\sqrt{L\lambda})$.

There are several open-loop schemes such as the harmonic broadcasting schemes [23, 25, 24], the permutation based pyramid scheme [1], and the Greedy Disk-Conserving Broadcast Scheme [17] which provide a $O(\text{Log}L)$ performance in terms of server bandwidth utilization. The open-loop schemes listed above also involve a non-zero delay between a client’s request and playout of the video clip. Paris et al.[26] propose a set of schemes involving *partial preloading* of certain objects to the set-top box of clients to eliminate this delay. The resulting protocols have zero delay but still involve substantial server bandwidth utilization when the request rates for the clips are relatively low.

The *catching* scheme discussed in [18] is a parameter-based scheme whose server channel utilization is $O(\text{Log}L\lambda)$, provided λ is accurately estimated. Thus, the parameters of the system can be chosen to provide better performance than the open-loop schemes when the request rate is not very high. However, a major disadvantage of this approach is that the performance of the system depends critically on a priori knowledge of the request, and the system parameters need to be altered as and when the request rate changes. This involves both monitoring the request rate and recalibration of parameters. The estimation of the arrival rate itself could be very complicated in many cases when arrivals are bursty.

Controlled multicast can be combined with catching to provide a “pseudo” adaptive protocol, called *selective catching* [18]. In selective catching, the bandwidth usage can be limited to $O(\text{Log}L\lambda)$ under high request rates, and to the same bandwidth usage as controlled multicast under low request rates. However, this scheme involves a priori knowledge of the arrival rate of requests to determine whether to use controlled multicast or catching. Consequently, it does not work well in environments where the future arrival rate of requests cannot be predicted.

Hierarchical multicast stream merging (HMSM [15, 14, 13]) is a family of closed loop schemes where the server bandwidth usage grows as $O(\text{Log}L\lambda)$.

Eager et al. [12, 11] proposed the first closed-loop scheme, called *Dynamic Skyscraper*, that can potentially provide $O(\text{Log}L)$ performance even under high request rates. The algorithm uses

segments whose lengths follow a fixed progression. Upon receiving a request from a client, the server schedules a fresh transmission of only some segments of the clip. The client obtains the remaining segments from transmissions which have already been scheduled but have not begun at the time of the request. However, there are potential conflicts in scheduling segments for transmission, and the authors do not provide a clear scheduling algorithm for the server regarding which segments it must retransmit when it receives a request from a client.

Besides this, this scheme still has a few disadvantages in that its design and performance are dependent on the minimum disk space available at a client. Under heterogenous environments where each client might have a different amount of disk space, it is difficult to know a priori the amount of disk space available in all clients. Furthermore, when this minimum disk space of clients is fairly small, its performance becomes much worse than that of open-loop schemes under almost all the request rates. Because of this, as the amount of video object that needs to be stored in regional caches increase, the tradeoffs between server bandwidth and cache bandwidth usages are not best exploited. Another disadvantage of the dynamic skyscraper scheme is the quantization into segments of fixed length. As a result, a client which requests a segment soon after the server has begun multicasting it requires a retransmission of the entire segment. The SMcache scheme proposed in this paper addresses this problem through the use of variable length patches.

3 The System Environment

Video-on-Demand systems normally consist of a set of servers which store video clips. The servers are connected to clients via high speed network. Each client is provided with some storage capability through its local disk. In addition, some clients are connected to a proxy server through the network.

It is shown in [8, 9] that the popularities of videos follow the Zipf distribution with the skew factor of 0.271, which means most of the demand (80%) is for a few (10 to 20) very popular video clips. Given the limited number of popular clips, it is possible for the proxy server to store the first few minutes or even seconds of each video clip, called the *prefix*. For each client that is not connected to a proxy server, we assume it has a local prefix cache with enough space to store the prefix of all the popular video clips. For simplicity, we refer to both proxy server and local prefix cache as *cache*.

The server stores video clips in their entirety. However only part of the clip after the prefix called the *body* need to be transmitted to the client upon request. Typically the server is provided with high bandwidth on its connection and clients have limited bandwidth devoted to streaming video clips. We assume the system has the following features:

- Each server may use an unlimited number of channels for transmitting a particular video object. This is a reasonable assumption to make, particularly when the server stores a very large number of video objects, and the total number of channels used by the server at any given instant is approximately equal to the mean, because of statistical multiplexing.

- Each client has some local disk space.
- In response to a request, the client first receives the prefix from the prefix cache.
- We assume that each client knows the location of its nearby cache and that the cache always contains the prefix of the video clip requested by the client.
- Because client bandwidth for streaming video clips is limited, the client may receive or record incoming transmissions from at most two channels at any given time. This include transmissions from the video servers and the cache.
- All channels have equal bandwidth. The transmission rate on all channels is constant and equal to the playout rate.
- Clients always request for the entire clip, from the beginning.

4 Multicast with Cache (Mcache)

This section describes the basic Mcache scheme. The video server multicasts the body of video clips using *object channels* and *patch channels*. Object channels are used to multicast the entire body of the video clip, whereas patch channels are used to multicast portions of the clip right after the prefix to facilitate late-arriving requests to catch up with a transmission on the object channel. In this algorithm, an object channel multicasts the entire body of the movie. It will be shown that because of this property, the algorithm requires the server to use $O(\sqrt{L})$ bandwidth on average. In Section 5, we extend this algorithm to allow the server to segment the body into multiple segments, and apply the basic Mcache algorithm on each segment independently to achieve $O(\log L)$ server bandwidth usage when the request rate is high.

4.1 The Mcache algorithm

The client's action is straightforward: it requests the clip body from the video server and requests the prefix from cache. The prefix is received immediately from cache and played out at the client. The server calculates a schedule and sends back to the client the information regarding the channels to join and the time to join each channel.

The server's multicast schedule is determined by the arrival time of a request and the status of existing multicast channels. If there is an ongoing multicast of the clip body on an object channel, depending on how long the multicast has been running, the server can choose to either (1) start multicasting a patch and instruct the client to join both the existing object channel and the patch channel, or (2) start a new multicast of the body on a new object channel (with no need for a patch channel if it receives from the beginning of an object channel). There is a tradeoff in this choice. If the multicast has not been running for a long time, then the patch would be short because the client has not missed much. Otherwise, the patch would be long. Thus, it might be more cost effective to receive the body from the beginning by starting a new object channel, rather than receiving a long patch. We apply a threshold of y time units, called *cutoff threshold*,

Server's Action:

```

Mcache-Server( $u, x, y, L$ )
Upon receipt of request at time  $u$  from client,
if (there is no mcast of the body in  $[u - y, u)$ ) and
   (there is no schedule of mcast of body in  $[u, u + x)$ )
then schedule a mcast of body at time  $u + x$ ;
(1) multicast body at time  $u + x$ ;
     $m =$  join mcast of body at time  $u + x$ ;
    send( $m$ , client);
elseif (there is no mcast of the body in  $[u - y, u)$ ) and
   (there is a scheduled mcast of body at  $t \in [u, u + x)$ )
(2)  $m =$  join mcast of body at time  $t$ ;
    send( $m$ , client);
elseif (there is a mcast of body that started at  $t \in [u - y, u)$ ) and
   (there is no patch scheduled in  $[u, \min(t + y, u + x))$ )
(3) schedule a mcast patch for  $[x, x + u + x - t]$ 
    at  $k = \min(t + y, u + x)$ ;
    multicast the patch for  $[x, x + u + x - t]$  at  $k$ ;
     $m =$  join mcast of patch at  $k$  and listen until  $k + u + x - t$ ;
    join mcast of body at  $u + x$  and listen until  $t + L$ ;
    send( $m$ , client);
elseif (there is a mcast of body that started at  $t \in [u - y, u)$ ) and
   (there is a patch scheduled at  $k \in [u, \min(t + y, u + x))$ )
(4) expand patch for range  $[x, x + u + x - t]$ 
     $m =$  join mcast of patch at  $k$  and listen until  $k + u + x - t$ ;
    join mcast of body at  $u + x$  and listen until  $t + L$ ;
    send( $m$ , client);
endif

```

Figure 1: Mcache Algorithm at the Server

to decide on the two cases. If the existing channel has been running more than y time units, then a new object channel is created. Otherwise, a patch is used.

To simplify the description, transmission and propagation delays are ignored. The algorithm can be easily modified taking these delays into account. We use constants x and L to denote the prefix length and the body length respectively, both in time units. The algorithm describes the actions taken by a client and the server when a client request comes in at time u . We denote the server algorithm by $Mcache(u, x, y, L)$. It takes as input the request time (u), the prefix length (x), the cutoff threshold (y), and the length of movie body (L). It schedules object channels or patch channels to handle the requests and returns the times that the client joins an object channel and a patch channel. Note that this algorithm is used as a subroutine for our main algorithm presented in Section 5. Below, we describe the server algorithm for Mcache. The pseudocode is shown in Figure 1.

Batching: When a client issues a request for the clip at time u , if there is no object channel that has started in $[u - y, u)$, or is scheduled to start in $[u, u + x)$, then the server schedules a new object channel at the latest possible time, which is $u + x$, and instructs the client to join at $u + x$ to get the entire body. Since the client receives the prefix from its cache during the first x time units, it can wait until $u + x$. Here, the threshold y is used to determine whether a new object channel or a patch has to be used. If there is an object channel scheduled to start in $[u, u + x)$, then the client simply joins this multicast when it starts.

Patching: When there is an object channel that started at $t \in [u - y, u)$, the client joins this

multicast at $u + x$. Since the multicast has already started, the client needs a patch for the first $u + x - t$ units of the clip body. This patch can also be multicast. The fact that the client joins the object channel at time $u + x$ instead of at u allows it to receive the patch on the second channel at any time in $(u, u + x]$ (recall that the client receives the prefix on one channel during $[u, u + x)$), which in turn facilitates batching together requests for the same patch. If there is already a patch channel scheduled to service earlier requests, then the client can join that channel when it starts. The patch length transmitted to that channel has to be extended up to $u + x - t$ to accommodate this client. This is how requests for a patch are batched together.

If no patch channel is scheduled to start before the client finishes receiving the prefix at the cache (it receives the prefix until time $u + x$), the server has to schedule one to start before $u + x$. However, this patch should start no later than $t + y$ because the existing object channel was started at t , and any request coming after y time later is serviced by a new object channel. So the starting time s of the patch is set to $\min\{y + t, u + x\}$. The patch consists of the first $u + x - t$ units of the clip body and may be extended later to accommodate future requests.

The threshold value y controls the frequency of recruiting a new object channel, and therefore affects average server bandwidth usage. Selecting too small a value for y results in a complete transmission of the clip body being scheduled too often. Selecting too large a value results in large patches which again require higher server bandwidth. This is similar in concept to the selection of an optimal patching threshold in Controlled Multicast [19].

4.2 Performance analysis of Mcache

In this section, we provide a discrete-time analysis to show how the mean server bandwidth is minimized by selecting the optimal y . The arrival of requests at the server is modeled by Poisson process with rate λ requests per time slot unit.

A client that requests for the video clip during time slot t initiates a new transmission if there is no new transmission that began after time slot $t - y - 1$. The new transmission is scheduled to start at the end of slot $t + x - 1$. If there is an ongoing transmission which began at the end of slot $s > t - y - 1$, the client joins the multicast at the end of slot $t + x$. It also receives the first $t + x - s$ slots of the body as a patch. This patch is scheduled for transmission at the end of slot $\min(s + y, t + x - 1)$.

Let τ_k be the time slot at the end of which the k^{th} complete transmission of the body of the video clip starts. Let n_k be the number of patches transmitted by the server in the interval $[\tau_k, \tau_{k+1})$. Let P_k be the total length of these n_k patches. We define $T_k = \tau_{k+1} - \tau_k$ to be the k^{th} *patching window*. The server begins a new full transmission of the body of the video clip at the start of every patching window. By applying renewal theory, it can be shown that P_k and T_k are i.i.d. random variables with means EP and ET respectively, and that the mean bandwidth requirement of the server is:

$$R_s = B \times \frac{L + EP}{ET} \quad (1)$$

ET is the mean length of a patching window. Any request arriving in the next $y - 1$ slots

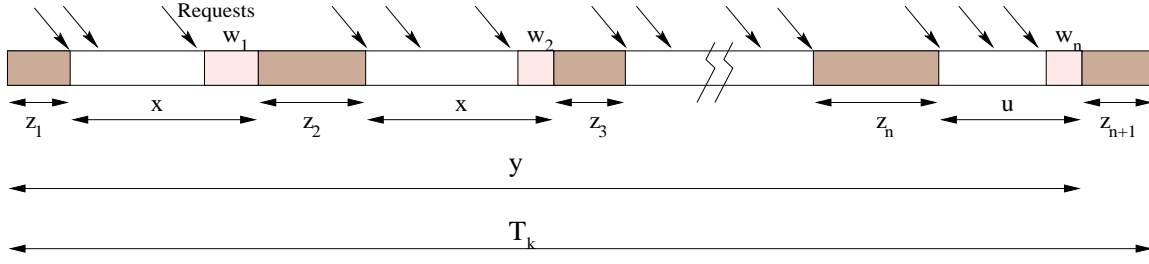


Figure 2: Sample Sequence of Requests to the Server

after a new transmission results in a patch. After the first y slots, any new request arriving in slot $y + u$, say, results in a new patching window that begins at the end of slot $y + u + x - 1$. Thus, the length of the patching window under consideration is $y + u + x - 1$ slots. Since request arrivals are Poisson, u is a geometric random variable with mean $\frac{1}{1-\alpha}$, where $\alpha = e^{-\lambda}$. Therefore, ET is equal to $y + x + \frac{\alpha}{1-\alpha}$.

To estimate EP , we first define the concept of a *patch segment* with the help of a sample sequence of requests in a typical patching window, and the corresponding patches generated, as shown in Figure 2.

Suppose there are no request arrivals in the first z_1 slots. The first request arrives in slot $z_1 + 1$. If $z_1 < y - x$, the server waits until time slot $z_1 + x$ before transmitting the patch for this client. When it does, the patch is multicast and must be sent to any client whose request arrived during or before slot $z_1 + x$.

We define the first $z_1 + x$ slots shown in Figure 2 to constitute *patch segment 1*, denoted by PS_1 . It consists of z_1 slots without any request, followed by a request in slot $z_1 + 1$, which is in turn followed by up to $x - 1$ slots which may feature further requests from other clients. The significance of PS_1 is that all clients which request the video clip during PS_1 receive the same patch; call it \mathcal{P}_1 . Suppose that the last such request arrived at $z_1 + x - w_1$. This request receives the prefix on one channel and the patch on the second channel until slot $z_1 - w_1 + 2x$, and receives the original multicast only after time slot $z_1 - w_1 + 2x$. The length of the first patch, \mathcal{P}_1 , is therefore given by $z_1 + 2x - w_1$.

Suppose there are no requests to the server from $z_1 + x$ to $z_1 + x + z_2$. The next request arrives during slot $z_1 + z_2 + x + 1$. If $z_1 + x + z_2 < y - x$, we call the segment constituting slots $z_1 + x + 1$ to $z_1 + z_2 + 2x$ as patch segment 2, or PS_2 . If $z_1 + z_2 + 2x \leq y$, all clients whose requests arrive during PS_2 receive the same patch, \mathcal{P}_2 . The corresponding patch is multicast by the server at $z_1 + z_2 + 2x$ and is of length $z_1 + z_2 + 3x - w_2$, where the last request in PS_2 arrives at $z_1 + z_2 + 2x - w_2$. Similarly, we can say that the j^{th} patch segment, PS_j consists of slots $z_1 + z_2 + \dots + z_{j-1} + (j-1)x + 1$ to $z_1 + z_2 + \dots + z_j + jx$. The length of the j^{th} patch is given by $p_j = \sum_{i=1}^j z_i + (j+1)x - w_j$, assuming $z_1 + z_2 + \dots + z_{j-1} + (j-1)x < y - x$.

The n^{th} patching segment PS_n consists of slots $z_1 + z_2 + \dots + z_{n-1} + (n-1)x + 1$ to $z_1 + z_2 + \dots + z_n + nx$. Assume that $z_1 + z_2 + \dots + z_{n-1} + nx < y$ but $z_1 + z_2 + \dots + z_n + nx \geq y$, i.e., there are exactly n patch segments in slots 1 to y . But observe that only requests that arrive during or prior to slot y are served by the n^{th} patch. Any requests arriving in the interval

$(y, z_1 + z_2 + \dots + z_n + nx]$ receive a new and complete transmission of the clip, since the cut-off parameter is y . Let $\psi_n = z_1 + z_2 + \dots + z_n + nx - y$.

We first estimate $\phi(n)$, the mean length of the first $n - 1$ patches, assuming that there are exactly n patch segments which begin at or before slot y .

$$\phi(n) = E_n \left\{ \sum_{j=1}^{n-1} (n-j)z_j + \frac{(n+2)(n-1)}{2}x - \sum_{j=1}^{n-1} w_j \right\} \quad (2)$$

THEOREM 1 z_1, z_2, \dots, z_{n-1} are identically distributed. Their mean, conditioned on having exactly n patch segments, is given by $EZ|_n \approx \frac{y+x/2}{n} - x$.

PROOF : See Appendix A.

The intervals w_1, w_2, \dots, w_{n-1} are also i.i.d. random variables independent of n . Let their mean be EW . $w_i, i < n$, are simply geometric random variables conditioned on $w_i < x$. Therefore, $EW = \frac{\alpha}{1-\alpha} - x \frac{\alpha^x}{1-\alpha^x}$. Thus, Equation (2) reduces to:

$$\phi(n) = \frac{n(n-1)}{2} \left(\frac{y+x/2}{n} - x \right) + \frac{(n+2)(n-1)}{2}x - (n-1)EW$$

On simplification, we get:

$$\phi(n) = \frac{y}{2}(n-1) + \left(\frac{5x}{4} - EW \right)(n-1) \quad (3)$$

We can un-condition on n to get

$$\Phi = \sum_n \phi(n) = \frac{y}{2}(EN-1) + \left(\frac{5x}{4} - EW \right)(EN-1)$$

where EN is the mean number of patches in interval T_k .

THEOREM 2 $EN \approx \frac{y(1-\alpha)}{x(1-\alpha)+1}$

PROOF : See Appendix B.

Let the mean length of the last patch in this interval is given by P_Ω .

THEOREM 3 $P_\Omega \approx y \frac{x(1-\alpha)}{x(1-\alpha)+\alpha}$

PROOF : See Appendix C.

Therefore, the mean bandwidth required by the server is given by $R_s = B \frac{L+\Phi+P_\Omega}{ET}$, which is approximately equal to:

$$B \left(\frac{L + \frac{y^2}{2(x+\frac{1}{1-\alpha})} + \frac{y}{2(x+\frac{1}{1-\alpha})} \left(\frac{7x}{2} - \frac{1}{1-\alpha} - 2EW \right) - \frac{3x}{2} + EW}{y + x + \frac{\alpha}{1-\alpha}} \right) \quad (4)$$

The optimal value of y is once again obtained by solving a quadratic equation. When $\lambda \rightarrow \infty$, the optimal value of y is given by $\sqrt{2Lx + x - \frac{11x^2}{2}} - x$. Therefore, the optimal value of R_s grows as $O(\sqrt{L})$. This indicates that the bandwidth usage is bounded by $O(\sqrt{L})$, independent of λ . The reason for this is that up to a certain value of λ , the server has to transmit more patches as λ increases. However, beyond this point, the server merely batches more requests on to a single patch, thus increasing multicast efficiency. This is made possible because of prefix caching.

5 Segmented Mcache (SMcache)

The Mcache algorithm described in the previous section outperforms Controlled Multicast (see [27]). This is easily seen as the outcome of batching all requests that arrive when a client is accessing the prefix of the clip. The hotter a given video clip gets, a larger number of requests are batched. However, the length of the cut-off is in the order of \sqrt{L} . This means the mean bandwidth required by the server is also $O(\sqrt{L})$, which is inferior to several known open loop schemes. For instance, the pre-loading schemes by Paris et al [26] can be used to provide instantaneous video on demand while using only $O(\text{Log}L)$ server bandwidth. In this section, we extend the basic Mcache algorithm to preserve the advantages of the original algorithm with nearly the same server bandwidth requirement as the open loop schemes under high request rates. This algorithm, presented below, is called *Segmented Mcache (SMcache)*.

5.1 The SMcache Algorithm

The body of the video clip is broken down into several segments. Let L be the length of the body and L_1, L_2, \dots, L_N be the lengths of segments 1, 2, ..., N , respectively. The first segment is transmitted according to the basic Mcache scheme discussed in the earlier section. Let x_1 be the length of the prefix, and y_1 be the cut-off parameter for segment 1. Assume that the server begins a full transmission of the first segment (i.e., the object channel for the first segment) at time 0. Consider a client which requests for the first segment at time u_1 , $u_1 < y_1$. The client accordingly receives the first $u_1 + x_1$ time units of the first segment as a patch, starting at time s . Note that s must be no later than $u_1 + x_1$ which is the time that the client finishes receiving the prefix of length x_1 . Further, s cannot be later than y_1 because any request coming after y_1 will be given a new object channel for the first segment. Thus, $s \leq \min(y_1, u_1 + x_1)$. The patch must last until time $s + u_1 + x_1$.

Observe that although the client may receive transmission from two channels at any given time, the client only listens to the patching channel until time $s + u_1 + x_1$. Note that the latest time this client can afford to delay receiving the second segment is $u_1 + L_1 + x_1$ because by this time, the first segment must have been fully played out. In other words, since the client leaves the patch channel at time $s + u_1 + x_1$, and it does not have to receive the next segment until $u_1 + L_1 + x_1$, the client listens to only one channel (the object channel for the first segment) during the interval $(s + u_1 + x_1, u_1 + L_1 + x_1)$. Since $s \leq y_1$, this interval includes the interval $(y_1 + u_1 + x_1, u_1 + L_1 + x_1)$, the duration of which is $L_1 - y_1$.

This is analogous to having a *virtual prefix* of length $L_1 - y_1$ for the second segment so that the server can delay serving the request for the second segment for up to a duration of length $L_1 - y_1$. To visualize this, suppose that the client whose original request arrived at u_1 makes a *virtual request* to the server for segment 2 at time $u_1 + y_1 + x_1$. The latest time until which the server may delay transmitting segment 2 to this client is actually $u_1 + L_1 + x_1$. Thus, the server has $L_1 - y_1$ time units after this virtual request to transmit this segment to the client or, if there is an ongoing transmission, to begin transmitting a patch to the client.

The above situation is the same as in Mcache when a prefix of length $x_2 = L_1 - y_1$ of a movie clip of length $L_2 + x_2$ is stored in the cache, and a client's request for that movie arrives to the Mcache server at time $u_1 + y_1 + x_1$. Now, consider the body of length L_2 with a prefix of length $L_1 - y_1$ being transmitted using the basic Mcache scheme. Assume that the cutoff threshold for the clip is y_2 . Also, consider a request for this clip arriving at time $s_1 = y_1 + u + x_1$. The client making this request receives the prefix on one channel during the interval $[s_1, L_1 - y_1 + s_1]$. Meanwhile, if there is a transmission of the clip body scheduled to begin in $[s_1, s_1 + L_1 - y_1)$, this client may join that multicast channel. If an existing object channel of the clip body began at time $k_2 \in [s_1 - y_2, s_1)$, the client joins this multicast at time $s_1 + L_1 - y_1$, and receives a patch of length $s_1 + L_1 - y_1 - k_2$ from the server. This is as if we run $Mcache(u_2, x_2, y_2, L_2)$ with $u_2 = u_1 + y_1 + x_1$, and $x_2 = L_1 - y_1$. This will give us the schedule for the client to get the second segment and its associated patch (if required). We can inductively apply the same argument for later segments; the schedule for segment $i + 1$ can also be obtained by running $Mcache(u_{i+1}, x_{i+1}, y_{i+1}, L_{i+1})$ ¹ with $u_{i+1} = u_i + y_i + x_i$, $x_{i+1} = L_i - y_i$, and y_i is the cutoff threshold for a movie length of L_i . (we discuss y_i for $i > 1$ in Section 6.1).

The SMcache server algorithm is shown in Figure 3.

Server's Action:

```

SMcache-Server( $u, x_1, y_1, L$ );
 $u_1 = u; S_L = 0; i = 1;$ 
while ( $S_L < L$ ) do
    Mcache( $u_i, x_i, y_i, L_i$ );
     $S_L = S_L + L_i;$ 
     $u_{i+1} = u_i + x_i + y_i;$ 
     $x_{i+1} = L_i - y_i;$ 
     $y_{i+1} = \min(\frac{L_i - y_i}{x_i} \times y_i, L_{i+1} - x_{i+1});$ 
     $i++;$ 
od

```

Figure 3: Algorithm at the Server

5.2 SMcache with Client Disk Limitations

So far, we assumed that the client can store up to half an entire video clip. Since the client may receive only on two channels at any given time, and the playout speed is assumed to be equal to the transmission speed, the client's disk space is never a limiting factor for the SMcache algorithm. Here, we take into account D_{max} , the disk space available at the client making a request, and modify the algorithm accordingly. The following constraints are introduced:

1. The client may begin receiving a segment at most the equivalent of D_{max} units of time before the scheduled playout time of the segment. Since $u_i + x_i + L_i$ is the scheduled playout time

¹We are extending and generalizing the notion of $Mcache(u_{i+1}, x_{i+1}, y_{i+1}, L_{i+1})$ because in Section 4, we presented $Mcache(u, x, y, L)$ in the present and past tenses. Here we have to run Mcache based on the future schedule. For instance, we say "If there is no object channel that has started in $[u - y, u)$, then the server schedules a new object channel at $u + x$." This has to be interpreted as "If there is no object channel that is scheduled to start in time $[u - y, u)$, then the server schedules a new object channel at $u + x$ ". The precise implementation of Mcache is in appendix.

```

Server's Action:
SMcache-Server( $u, x_1, y_1, L$ );
 $u_1 = u; S_L = 0; i = 1$ ;
while ( $S_L < L$ ) do
    Mcache( $u_i, x_i, y_i, L_i$ );
     $S_L = S_L + L_i$ ;
     $u_{i+1} = \max(u_i + x_i + y_i, u_1 + x_1 + S_L - D_{max})$ ;
     $x_{i+1} = \min(L_i - y_i, D_{max})$ ;
     $y_{i+1} = \min(\frac{L_i - y_i}{x_i} \times y_i, \min(L_{i+1}, D_{max}) - x_{i+1})$ ;
     $i++$ ;
od

```

Figure 4: SMcache with limited client disk space

for segment $i + 1$, and u_{i+1} is the instant of the virtual request for segment $i + 1$, u_{i+1} must be larger than or equal to $u_i + x_i + L_i + D_{max}$.

2. The maximum patch size for segment i may not exceed D_{max} . Thus, $y_i + x_i \leq D_{max}$.

With these constraints, we rewrite the server algorithm in Figure 4.

5.3 Partitioned SMcache

In this section, we consider a generalized version of SMcache, in which the proxy servers store a few of the initial segments of the video object in addition to the prefix. This is on similar lines as the partitioned dynamic skyscraper model proposed in [11]. We assume there is one main server and M regional caches in the network. Each regional cache stores the first n segments of the body of the clip, in addition to the prefix of length x_1 . The main server stores the remaining $N - n$ segments. The SMcache server algorithm is suitably modified to facilitate a regional cache to multicast the first n segments to clients in its network neighbourhood and the main server to transmit the latter $N - n$ segments. As before, the regional cache transmits the prefix to a client immediately upon receiving a request. In addition, the basic Mcache algorithm is executed at the proxy server only for those segments stored in cache. The main server and proxy server algorithms are as shown in Figure 5.

6 Performance Analysis

In this section, we show that the mean number of server channels used by the non-partitioned SMcache algorithm can be upper-bounded by a function which is $O(\text{Log}L)$, provided it is not limited by disk capacity at the clients. Then, we prove that the server bandwidth usage goes to zero when the request rate goes to zero. Finally, we analyze the performance of the partitioned SMcache scheme and develop an optimization model for partitioning a video clip between the main server and regional caches.

```

Proxy Server's Action:
Part-SMcache-Proxy-Server( $u, x_1, y_1, L, L_1, n$ );
 $u_1 = u$ ;
Send-prefix( $time = u_1, length = x_1$ );
for ( $i = 1$  to  $n$ ) do
    Mcache( $u_i, x_i, y_i, L_i$ );
     $u_{i+1} = u_i + x_i + y_i$ ;
     $x_{i+1} = L_i - y_i$ ;
     $y_{i+1} = \min(\frac{L_i - y_i}{x_i} \times y_i, L_{i+1} - x_{i+1})$ ;
od

Main Server's Action:
Part-SMcache-Main-Server( $u, x_1, y_1, L, L_1, n$ );
 $u_1 = u$ ;
 $S_L = 0$ ;
 $i = 1$ ;
while ( $S_L < L$ ) do
    if ( $i > n$ ) Mcache( $u_i, x_i, y_i, L_i$ );
     $S_L = S_L + L_i$ ;
     $u_{i+1} = u_i + x_i + y_i$ ;
     $x_{i+1} = L_i - y_i$ ;
     $y_{i+1} = \min(\frac{L_i - y_i}{x_i} \times y_i, L_{i+1} - x_{i+1})$ ;
     $i++$ ;
od

```

Figure 5: The Partitioned SMcache algorithms

6.1 An Upper Bound Analysis of SMcache

Suppose we choose L_1 and y_1 such that $x_2 = L_1 - y_1 > x_1$. Let us assume that $x_2 = \beta x_1$, where $\beta > 1$. Therefore, the mean bandwidth $R_s^{(1)}$, required by the server in transmitting segment 1 may be estimated from (4). Rearranging the terms, it can be shown that:

$$R_s^{(1)} < B \frac{L_1 + \frac{y_1^2}{2x_1} + 2y_1 + x_1}{y_1} = B_{max} \quad (5)$$

Now, let us choose $L_2 = \beta L_1$ and $y_2 = \beta y_1$. Then, we can show that:

$$R_s^{(2)} < B \frac{L_2 + \frac{y_2^2}{2x_2} + 2y_2 + x_2}{y_2} = B \frac{L_1 + \frac{y_1^2}{2x_1} + 2y_1 + x_1}{y_1} = B_{max}$$

Since $\frac{L_2}{L_1} = \frac{x_2}{x_1} = \frac{y_2}{y_1} = \beta$, we have $x_3 = L_2 - y_2 = \beta(L_1 - y_1) = \beta x_2$. Thus, we choose each segment to be larger than the previous by a factor of β . The final segment L_N is the only exception, and it is given by:

$$L_N = L - \sum_{i=1}^{N-1} L_i \leq \beta L_{N-1}$$

Since $\frac{L_N}{x_N} \leq \frac{L_{N-1}}{x_{N-1}} = \beta$, it can be shown that $R_s^{(N)} \leq R_s^{(N-1)} < B_{max}$. Thus, the total bandwidth required by the server in transmitting all N segments is $R_s < N B_{max}$. But $L_N = L - \sum_{i=1}^{N-1} L_i > 0$. Therefore :

$$L > \sum_{i=1}^{N-1} L_i = \sum_{i=0}^{N-2} L_1 \beta^i$$

Or, $L_1 \frac{\beta^{N-1} - 1}{\beta - 1} < L$, which gives us

$$N < 1 + \text{Log}_\beta \left(\frac{L(\beta - 1) + L_1}{L_1} \right)$$

Therefore, we have

$$R_s < NB_{max} < B_{max} \left[1 + \text{Log}_\beta \left(\frac{L(\beta - 1) + L_1}{L_1} \right) \right] \quad (6)$$

From (6), it can be seen that the required bandwidth is no greater than $O(\text{Log}L)$.

6.2 The Closed-Loop Advantage of SMcache

Rearranging terms in Equation (4), we can show that:

$$R_s^{(1)} < B(1 - \alpha) \left(L_1 + \frac{y_1^2}{2x_1} + 2y_1 + x_1 \right) = (1 - \alpha)B_0$$

We also have:

$$R_s^{(2)} < B(1 - \alpha) \left(L_2 + \frac{y_2^2}{2x_2} + 2y_2 + x_2 \right) = (1 - \alpha)B_0 \times \beta$$

In general, we have:

$$R_s^{(i)} < B(1 - \alpha) \left(L_i + \frac{y_i^2}{2x_i} + 2y_i + x_i \right) = (1 - \alpha)B_0 \times \beta^{i-1}$$

From this, we can show that:

$$R_s = \sum_{i=1}^N R_s^{(i)} < \sum_{(i=1)}^N (1 - \alpha)B_0 \times \beta^{i-1} = B_0 \left(\frac{\beta^N - 1}{\beta - 1} \right) (1 - \alpha) \quad (7)$$

But $\alpha = e^{-\lambda}$. From (7), we see that when $\lambda \rightarrow 0$, $R_s < B_0 \left(\frac{\beta^N - 1}{\beta - 1} \right) (1 - \alpha) \rightarrow 0$. In other words, the bandwidth usage of the new scheme is not only bounded by an $O(\text{Log}(L))$ function, but is also very small when the request rate is small (i.e., proportional to $1 - e^{-\lambda}$). Open-loop schemes such as periodic broadcast, on the other hand, requires a fixed server bandwidth irrespective of request rates. This is the deciding advantage in using a closed loop scheme over even the most efficient open-loop schemes.

6.3 Performance of Partitioned SMcache

In this section, we consider the performance of partitioned SMcache where, in addition to the prefix, some segments of the movie body are moved into the proxy server. We discuss an optimization model based on network costs for transmitting a video object as well as storage costs for replicating these segments at the regional caches. The solution to this optimization model provides the optimal regional caching strategy, similar to the one discussed in [11].

Let λ_m , $m = 1, 2, \dots, M$, be the mean request rate generated by clients in the network neighbourhood of regional cache m . Then, the mean bandwidth required at regional cache m is given by:

$$R_c(\lambda_m) = B(\lambda_m x_1) + \sum_{k=1}^n R_c^{(k)}(\lambda_m)$$

where $R_c^{(k)}(\lambda_m)$ is the mean bandwidth required by the regional cache in transmitting segment k to its neighbouring clients, calculated as given by (4). The mean bandwidth requirement at the main server is given by:

$$R_s(\sum_m \lambda_m) = R_s(\lambda) = \sum_{k=n+1}^N R_s^{(k)}(\lambda)$$

where $R_s^{(k)}(\lambda)$ is the mean bandwidth required by the main server in transmitting the k^{th} segment to all the clients. Obviously, moving more segments out to the regional caches results in a reduction in the network load at the main server and an increase at the regional caches.

We note that it may be preferable to increase the network load at the regional caches if it results in decreasing the load at the main server, for purposes of even distribution of network traffic. However, there is another cost involved in increasing the fraction of the video object that is cached, viz., the cost of replicating more data and storing it in several locations. If there is no cost involved in replication and storage, the simple solution would be to store the entire video object at each regional cache. This not only helps in distributing the load evenly over the network, but also reduces the overall network load if the choice of cache location is made carefully. However, when there is a cost incurred in replication and storage of the leading segment set, we can devise an optimization model to arrive at the best caching strategy.

Let θ_m , $m = 1, 2, \dots, M$, be the time average cost incurred by regional cache m for transmitting on one channel. Therefore, the network cost for regional cache m is given by $\theta_m R_c(\lambda_m)$. Let Θ be the corresponding cost for the main server. Without loss of generalization, we can assume $\Theta = 1$. The network cost for the main server is $R_s(\lambda)$. Let the cost per unit time of storing a unit of data at regional cache m be ω_m . The storage (or replication) cost at regional cache m is then given by $\omega_m \sum_{k=1}^n L_k$. Let the cost per unit time of storing a unit of data at the main server be Ω . The storage cost at the main server is $\Omega \sum_{k=n+1}^N L_k$. The total cost per unit time for the SMcache system is given by:

$$C_T = R_s(\lambda) + \sum_{k=n+1}^N \Omega L_k + \sum_m \left(R_c(\lambda_m) \theta_m + \omega_m \left(x_1 + \sum_{k=1}^n L_k \right) \right) \quad (8)$$

We expect θ_m to be less than Θ because the main server multicasts to a larger number of clients spread out through the network whereas a regional cache multicasts to a relatively small number of clients in its own neighbourhood. For instance, consider a binary tree kind of network topology. The main server is at the root of the tree and there are 2^K clients are at the leaves. Assume all regional servers are at the intermediate nodes, and at the same level. Thus, a multicast by the main server can be received by up to 2^K clients, and traverses up to $2^{K+1} - 1$ branches. Each regional cache, on the other hand, only needs to multicast to at most $2^{K - \text{Log}_2 M}$ clients, while using only up to $2^{K - \text{Log}_2 M + 1} - 1$ branches. This indicates that there may be an advantage in moving more segments to the regional caches. However, this results in an increase in the storage costs (or replication costs) of more segments at each regional cache. Thus there is a trade-off between network and storage costs.

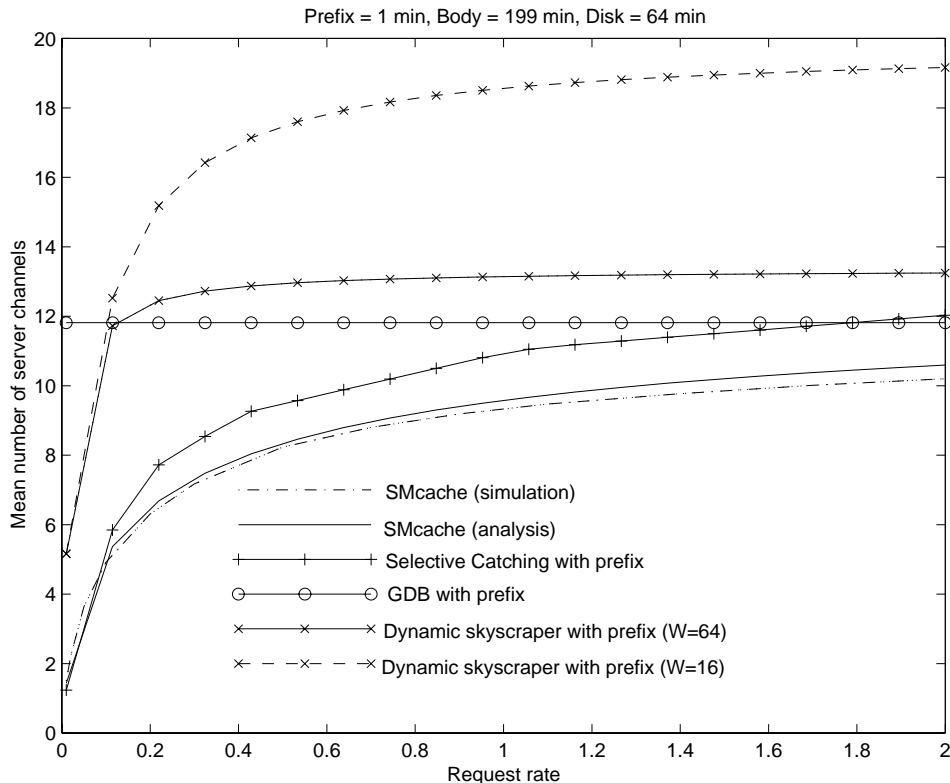


Figure 6: Server channel usage vs λ

7 Performance Comparison

In this section, we compare the performance of the SMcache scheme with that of dynamic skyscraper, GDB, and selective catching, which represent the state of the art in closed-loop, open-loop and hybrid VoD algorithms respectively. We modified each of these schemes to include a prefix cache with prefix size equal to that used by the SMcache scheme, so as to allow zero-delay with increased batching. The prefix is stored at proxies and transmitted by unicast to a client immediately upon receiving a request from that client. First, we study the performance of non-partitioned SMcache for various arrival rates of requests, when the request arrival process is bursty, when the disk space at clients is variable, and for various prefix lengths. Then, we study the performance of partitioned SMcache. Specifically, we study how the performance of the main server and that of the regional caches as functions of the number of segments stored in the regional caches. We compare these results to the performance of the main server and regional caches in the partitioned dynamic skyscraper algorithm, assuming that the same fraction of the video object is stored in the regional caches. Finally, we consider a sample cost function specified in (8) and plot the total network and storage costs of each algorithm. We also plot the total cost at the main server and at the regional caches to highlight the trade-off involved in transferring more data to the regional cache.

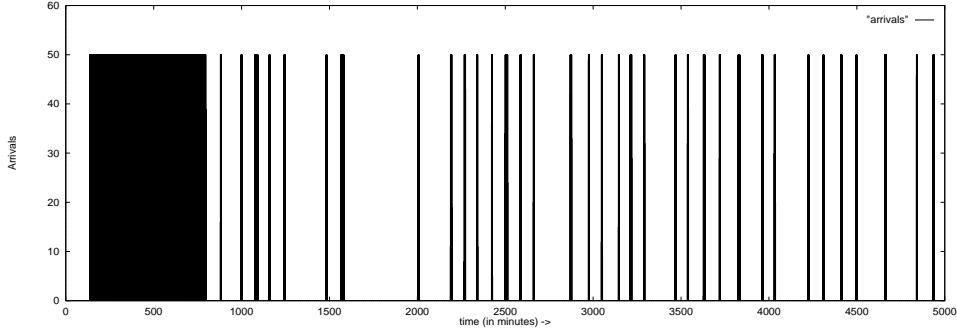


Figure 7: The Pareto Arrival process

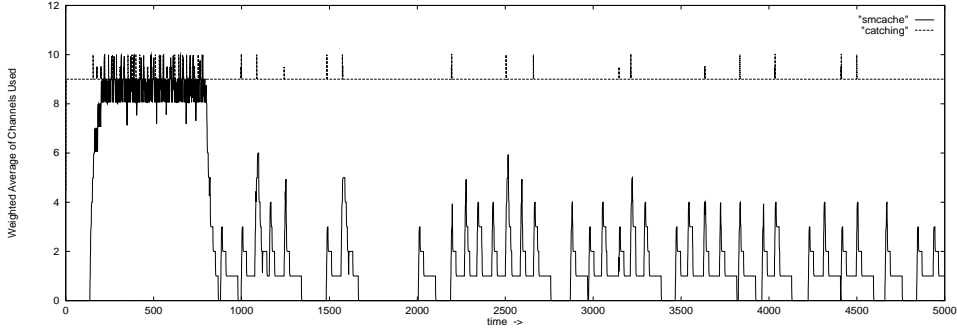


Figure 8: SMcache vs Selective Catching

7.1 Non-partitioned SMcache

Server bandwidth vs request rate: In Figure 6, we plot the average number of server channels required as a function of the request arrival rate, assuming Poisson request arrivals. We show the results for SMcache obtained by analysis as well as simulation. We assumed that each client has enough disk space to accommodate at least 64 minutes of the clip. dynamic skyscraper was evaluated for two values of W which represents the maximum segment length allowed by the algorithm, i.e., for $W = 16$ minutes and $W = 64$ minutes. First of all, we conclude that the approximate analytical algorithm provided by us to evaluate this algorithm for Poisson requests is very accurate. Next, we conclude that SMcache has considerable improvement over GDB when the request rate is low. We also conclude from Figure 6 that SMcache provides a significant improvement over dynamic skyscraper, while using the same resources. Observe that the choice of W is crucial to the performance of the dynamic skyscraper. In order to provide zero-delay VoD, W must be chosen to accommodate the client with the least disk space, so even if a small fraction of clients have the equivalent of 16 minutes of disk space instead of 64, W needs to be chosen accordingly. More importantly, the server needs to know the disk space available at each client in advance so as to design the segment lengths in correspondence. Finally, we conclude that optimal selective catching performs no better than SMcache. However, selective catching can be optimized only for a particular request arrival rate. SMcache, on the other hand, does not rely on a priori knowledge of the request arrival rate and easily adapts to changing request rates.

Server bandwidth for bursty arrivals: To illustrate the adaptiveness of SMcache, we

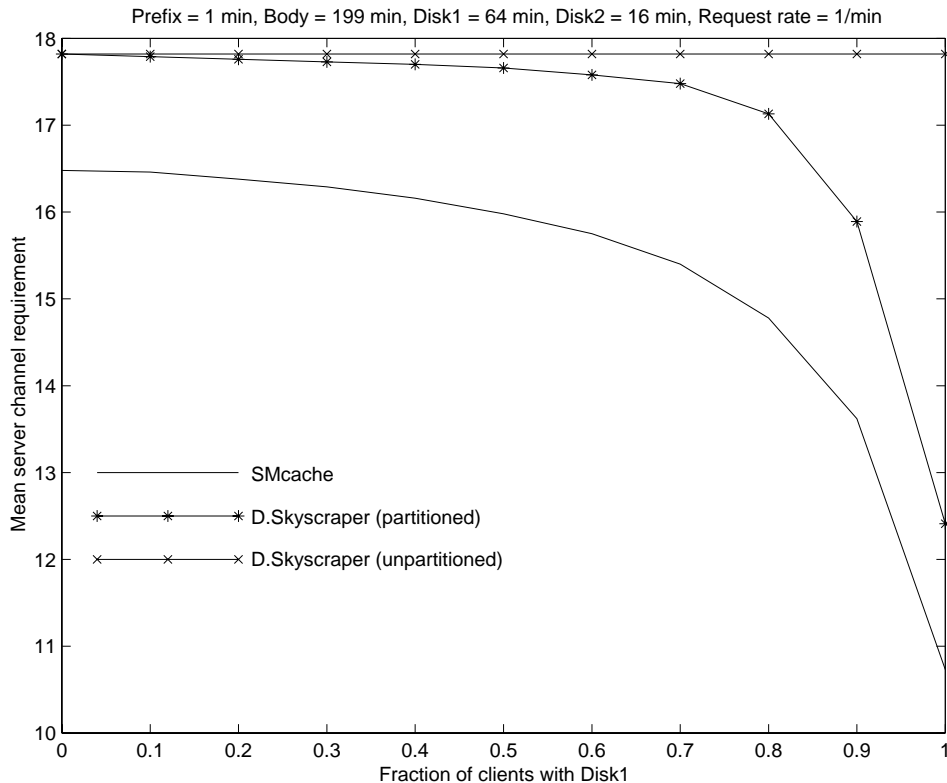


Figure 9: SMcache with Heterogeneous Clients

compare the performance with Pareto arrivals. Figure 8 contains the short-term channel usages of SMcache and selective catching (with prefix cache) as a function of time. These results were obtained by simulation. The Pareto arrival process is a bursty arrival process with long-range dependence, as shown in Figure 7. When the selective catching scheme is optimized for the mean arrival rate, there is a significant residual bandwidth when there are no arrivals. Besides, it requires the server to know the mean request rate in advance. With SMcache, there is no residual bandwidth, and there is no significant drop in performance due to erroneous estimation of the request rate.

Server bandwidth with limited client disk: Next, we illustrate the versatility of the SMcache algorithm in adapting to clients with variable disk space. In Figure 9, we compare the performance of SMcache with that of dynamic skyscraper (with prefix cache) when clients have varying disk space. Some clients have 16 minutes of disk space while the rest have 64 minutes. Thus, dynamic skyscraper algorithm needs to fix W at 16 minutes. We consider two versions of dynamic skyscraper. In the non-partitioned version, only the prefix is stored in cache. In the partitioned version, in addition to the prefix, a few initial segments (called leading segments) are stored in cache. We obtained the number of server channels used by SMcache and non-partitioned dynamic skyscraper through simulation. We add this to the mean number of proxy channels used given by $\lambda \times x_1$.

For partitioned dynamic skyscraper, we obtained the sum of the channels used by both the main server and the proxy server, assuming that there is a single proxy server. We plot the

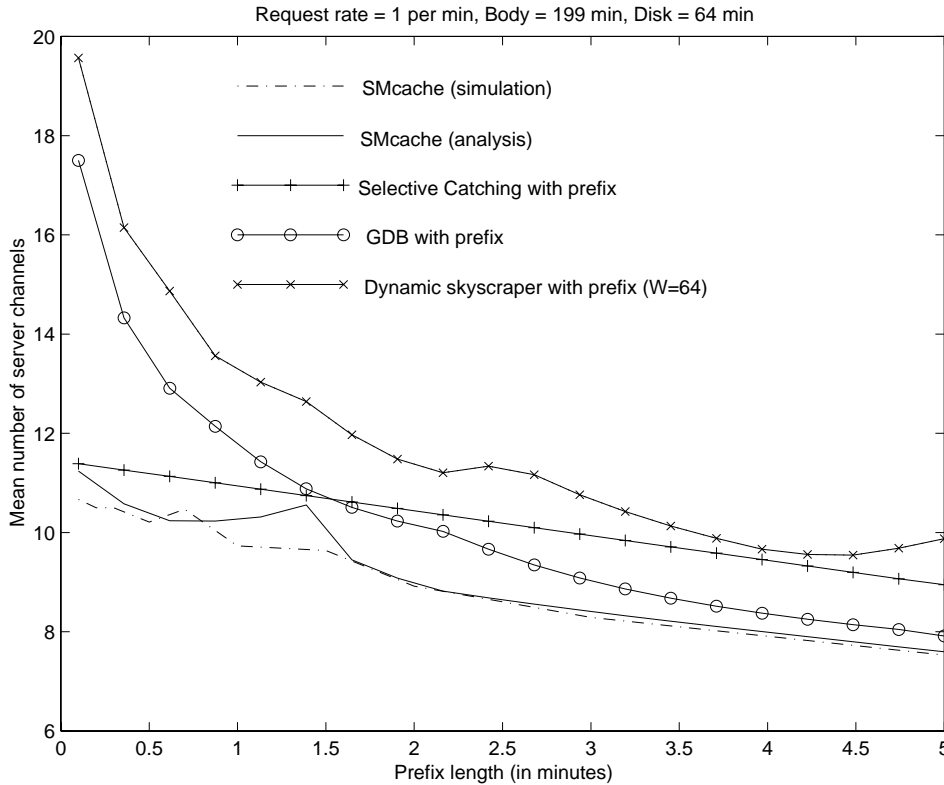


Figure 10: Server load vs Prefix Size

results as a function of the fraction of clients whose disk can store 64 minutes of video. Observe from Figure 9 that SMcache easily outperforms non-partitioned dynamic skyscraper which is not adaptive to clients with varying storage capacity. SMcache is also better than the partitioned version. It is to be noted that the partitioned version requires replication of resources at local (or regional) servers, and thus involves additional storage cost than SMcache, with no benefit. For instance, the partitioned dynamic skyscraper in the above example requires storing 63 minutes of the video in the proxy server. On the other hand, SMcache needs to store only 1 minute at the proxy server and still shows better performance. Furthermore, SMcache does not require proxy servers to have multicast capability, unlike partitioned dynamic skyscraper.

SMcache performance vs prefix length: Finally, we study the impact of prefix length on the performance of SMcache. In Figure 10, we compare the performance of the various schemes for different prefix sizes. When the prefix is large, the main server requires fewer channels for transmitting the video clip, but the network load at the cache increases. Since we assume that caches transmit the prefix by unicast, the mean number of channels required by a proxy server is given by the length of the prefix times the arrival rate of requests to that particular proxy. Therefore the sum of the mean number of channels required by all proxies is given by the total request rate at the main server, i.e., λ , times the prefix length. In Figure 10, we plot the mean number of channels required by the main server as a function of the prefix size. The values for SMcache were obtained both by simulation and analysis, and we observe that the results obtained by both methods are in agreement with each other. From the figure, we infer that

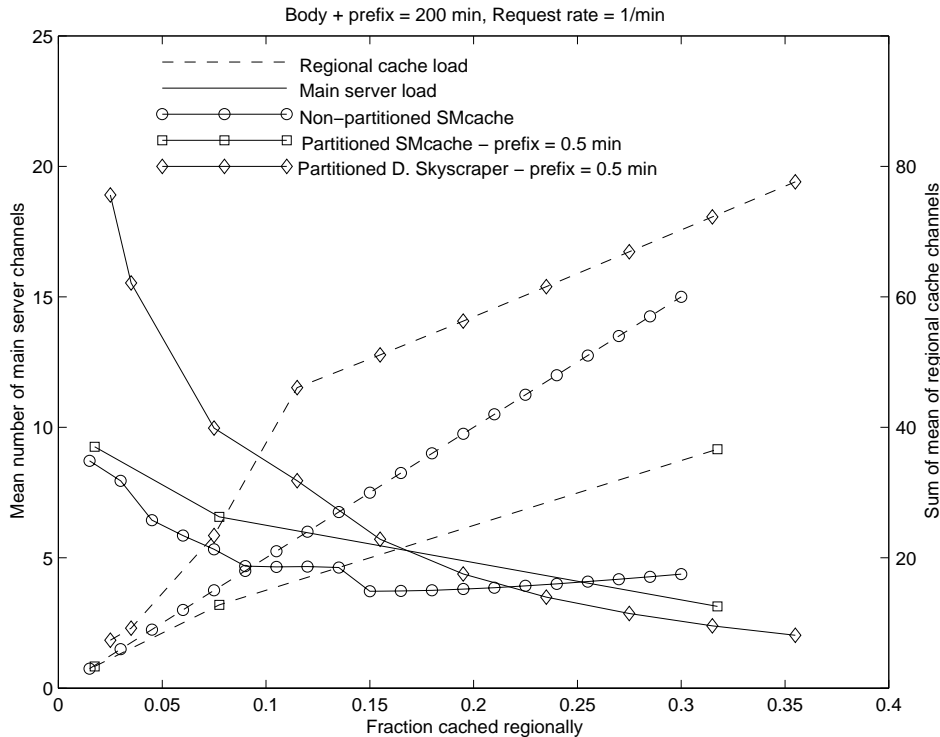


Figure 11: Partitioned SMcache: Network load at Main server and Regional cache

SMcache provides better performance than every other scheme for the parameters considered. Furthermore, for relatively small prefix lengths (less than one minute in Figure 10), SMcache shows a major improvement over prefixed dynamic skyscraper and prefixed GDB. This illustrates the major gains that could be had in SMcache even when the available cache space is very small, which are notably absent in dynamic skyscraper.

Obviously, there is a trade off involved in choosing the length of the prefix. A small prefix results in greater network load at the server whereas a longer prefix reduces server load but increases the channel requirement at the proxies. If the ratio β is fixed, then it can be shown that the optimal length of the prefix is given by: $x_{opt} \approx \frac{1}{\beta\lambda\theta}$, where θ is the ratio of the network cost of one client channel to a server channel.

7.2 Partitioned SMcache

Network load at server and cache: In Figure 11, we plot the number of channels required at the main server in partitioned SMcache and dynamic skyscraper, as a function of the fraction of the video object stored in the cache. Using dashed lines, we also plot the sum of the mean number of channels required by each regional cache, whose values correspond to the scale on the right of the graph. We also analyzed the non-partitioned SMcache algorithm assuming that the same fraction of the video clip was cached at regional servers as the prefix. For the parameters chosen, partitioned SMcache had a much lower regional cache load than partitioned dynamic skyscraper, without any significant deterioration in the load at the main server. We also infer that partitioned SMcache is better when we need to cache a larger fraction of the object regionally.

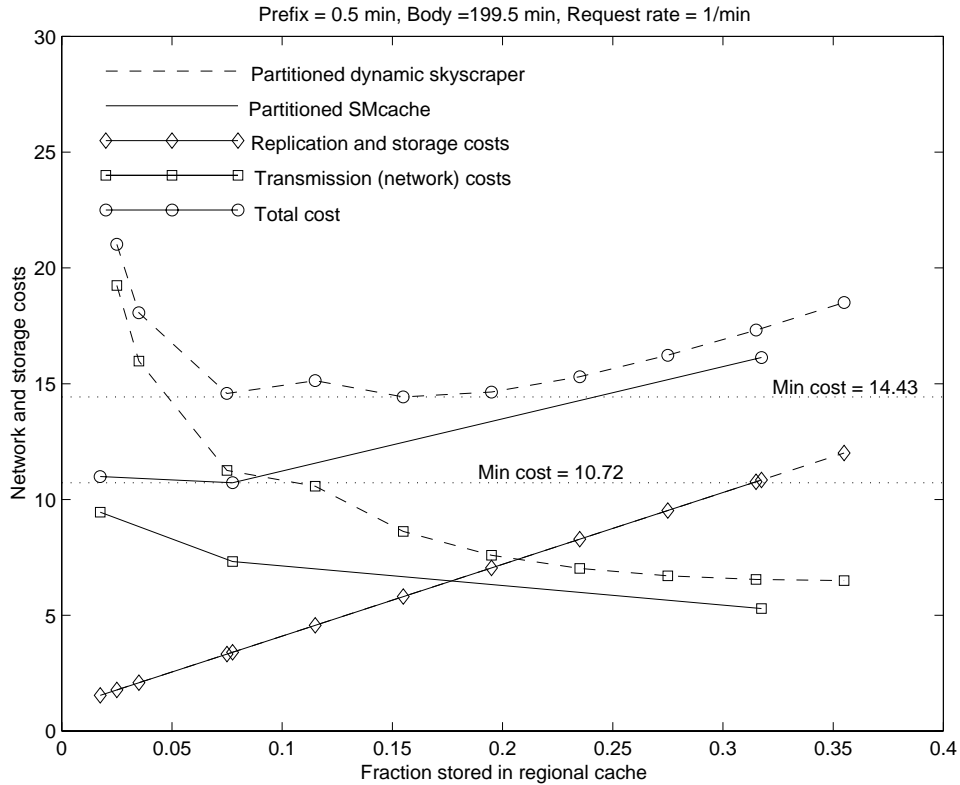


Figure 12: Transmission and Replication costs

Optimal Partitioning: Figure 12 illustrates this trade-off, as we plot the replication and transmission costs as a function of the fraction of the video object that is stored in the cache. As the fraction stored in cache increases, the total network cost can be expected to decrease if the location of the regional cache is chosen carefully. However, the replication cost increases. We have shown the cost for both partitioned dynamic skyscraper and partitioned SMcache. The cost parameters were chosen as follows: $M = 16$, $\Theta = 1$, $\theta = 0.059$, $\omega = 0.01$, and $\Omega = 0.005$. Correspondingly, the optimal cached fraction for SMcache is about 0.07, whereas it is about 0.15 for dynamic skyscraper. The minimum cost for SMcache is about 10.72, i.e., about 35% better than dynamic skyscraper. Figure 13 represents the trade-off between the total cost (i.e., network and storage costs) incurred by the main server versus that of the regional cache as we vary the fraction of the video object that is cached.

From the above curves, we can find the optimal partition. However, we further simplify this for some special cases as follows. Assuming that each segment is played out at the maximum rate whether is located at a regional cache or at the main server, and that $\theta_m = \theta$ and $\omega_m = \omega$, the optimum number of segments which need to be stored in cache can be shown to be: $\mathbf{n}_{\text{opt}} \leq \log_{\beta} \left(\frac{1-M\theta}{L_1(M\omega-\Omega)} \right) \leq \mathbf{n}_{\text{opt}} + 1$

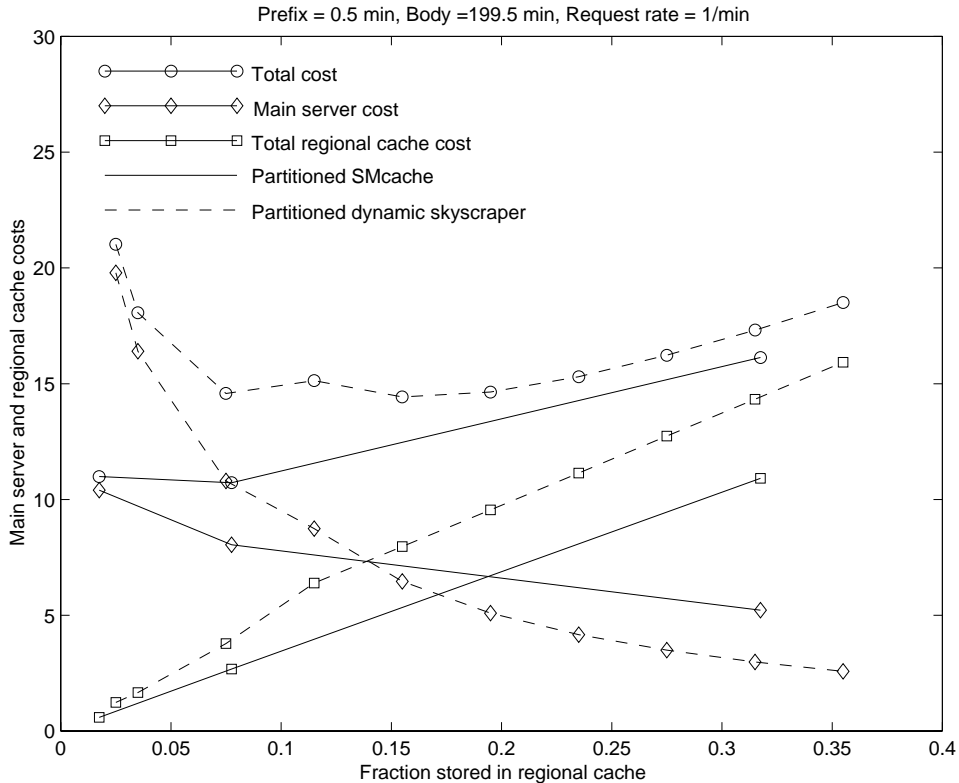


Figure 13: Main server and Regional cache costs

8 Conclusions

In this paper, we have presented a closed-loop scheme called Mcache, for providing zero-delay video-on-demand services. The use of prefix cache is critical to the efficient utilization of server bandwidth in these schemes. This is due to the fact that prefix caching allows batching of requests from different clients for a given video clip, while still providing zero-delay service.

The SMcache protocol is a generalized and improved version of Mcache where the clip is partitioned into several segments in order to exploit the availability of two receiving channels at each client to a greater extent than Mcache does. SMcache shows a marked improvement over Mcache in terms of server bandwidth usage when the prefix ratio is large, i.e., when the clips are large or the prefix is relatively small. Furthermore, SMcache limits the server bandwidth requirement to $O(\text{Log}(L))$, where L is the length of the body of the clip. This is the same as in open-loop schemes such as periodic broadcast schemes. However, SMcache being closed-loop, the server bandwidth usage is lower than periodic broadcast when the request rates are low.

Both Mcache and SMcache are adaptive in that the mean transmission rate of the server is altered according to the transient request rate for any given clip. This is a notable improvement of the caching scheme which is “pseudo-adaptive” in the sense that it involves recalibration of parameters whenever request rates change.

SMcache is also adaptive to varying disk space among clients. This is a distinct advantage over the dynamic skyscraper protocol whose design is dependent on the minimum disk space

available at a client. In addition to this, our experiments show that SMcache has significantly better performance than dynamic skyscraper when the prefix is very small in relation to the length of a clip.

References

- [1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. A permutation-based pyramid broadcasting scheme for video-on-demand systems. In *Proc. of IEEE International Conference on Multimedia Computing and Systems*, pages 118–126, Jun 1996.
- [2] C.C. Aggarwal, J.L. Wolf, and P.S. Yu. On optimal batching policies for video-on-demand storage servers. In *Proc. of International Conference on Multimedia Systems'96*, June 1996.
- [3] H. C. De Bey. Program transmission optimization, Mar 1995.
- [4] Y. Birk and R. Mondri. Tailored transmissions for efficient near-video-on-demand service. In *Proc. of IEEE International Conference on Multimedia Computing and Systems*, Florence, Italy, Jun 1999.
- [5] Y. Cai, K. A. Hua, and K. Vu. Optimizing patching performance. In *Proceedings of IS&T SPIE Conference on Multimedia Computing and Networking 1999 (MMCN '99)*, San Jose, California, 1999.
- [6] T. Choi, Y. Kim, and K. Chung. A prefetching scheme based on analysis of user access pattern in news-on-demand system. In *Proc. of the Seventh ACM International Multimedia Conference*, Oct 1999.
- [7] A. Dan, Y. Heights, and D. Sitaram. Generalized interval caching policy for mixed interactive and long video workloads. In *Proc. of SPIE's Conference on Multimedia Computing and Networking*, pages 344–351, San Jose, CA, Jan 1996.
- [8] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proc. of ACM Multimedia Conference*, Oct 1994.
- [9] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *Multimedia Systems*, 4(3):51–58, June 1996.
- [10] S. Deing. Empirical model of www document arrivals at access link. In *Proc. of the 1996 IEEE International Conference on Communication*, Jun 1996.
- [11] D. L. Eager, M. C. Ferris, and M. K. Vernon. Optimized regional caching for on-demand data delivery. In *Proc. of Multimedia Computing and Networking (MMCN'99)*, San Jose, California, Jan 1999.
- [12] D. L. Eager and M. K. Vernon. Dynamic skyscraper broadcasts for video-on-demand. In *Proc. of the 4th International Workshop on Multimedia Information Systems (MIS'98)*, Istanbul, Turkey, Sept 1998.
- [13] D. L. Eager, M. K. Vernon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. In *Proceedings of the 5th International Workshop on Multimedia Information Systems (MIS '99)*, Indian Wells, California, 1999.
- [14] D. L. Eager, M. K. Vernon, and J. Zahorjan. Optimal and efficient merging schedules for video-on-demand servers. In *Proceedings of the 7th ACM International Multimedia Conference (ACM Multimedia '99)*, Orlando, Florida, 1999.
- [15] D. L. Eager, M. K. Vernon, and J. Zahorjan. Bandwidth skimming: A technique for cost-effective video-on-demand. In *Proceedings of IS&T SPIE Conference on Multimedia Computing and Networking 2000 (MMCN 2000)*, San Jose, California, 2000.
- [16] A. Feldmann. *Modelling characteristics of TCP connections*. Technical Report, AT&T Laboratories, 1996.
- [17] L. Gao, J. Kurose, and D. Towsley. Efficient schemes for broadcasting popular videos. In *Proc. of NOSSDAV*, Cambridge, UK, Jul 1998.
- [18] L. Gao, J. Kurose, and D. Towsley. Catching and selective catching: Efficient latency reduction techniques for delivering continuous multimedia streams. In *Proc. of ACM Multimedia Conference*, Oct.-Nov. 1999.
- [19] L. Gao and D. Towsley. Supplying instantaneous video-on-demand services using controlled multicast. In *Proc. of IEEE International Conference on Multimedia Computing and Systems*, Florence, Italy, Jun 1999.
- [20] L. Golubchik, L. C.-S. Liu, and R.R. Muntz. Reducing (i/o) demand in video-on-demand storage servers. In *Proc. of ACM SIGMETRICS*, pages 25–36, 1995.

- [21] K. A. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true video-on-demand services. In *Proc. of ACM Multimedia Conference*, Bristol, England, Sept 1998.
- [22] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *Proc. of ACM SIGCOMM*, Sept 1997.
- [23] L-S. Juhn and L-M. Tseng. Harmonic broadcasting for video-on-demand service. *IEEE Transaction on Broadcasting*, 43(3):268–271, Sept 1997.
- [24] J-F. Paris, S. W. Carter, and D. D. E. Long. A low bandwidth broadcasting protocol for video on demand. In *Proc. of the 7th International Conference on Computer Communications and Networks*, pages 690–697, Oct 1998.
- [25] J-F. Paris, S. W. Carter, and D. D. E. Long. Efficient boradcasting protocols for video on demand. In *Proc. of the 6th International Symposium on Modeling, Analysis and Simulation of Computere and Telecommunication systems*, pages 127–132, July 1999.
- [26] J-F. Paris, D. D. E. Long, and P. E. Mantey. Zero-delay broadcasting protocols for video-on-demand. In *Proc. of 7th ACM International Multimedia Conference*, Oct-Nov 1999.
- [27] S. Ramesh and I. Rhee. *Multicast with Cache (Mcache): An Adaptive Zero Delay Video on Demand Service*. Technical Draft, Department of Computer Science, North Carolina State University, URL=<http://www.csc.ncsu.edu/~rhee/WWW/export/mcache.ps>, 1999.
- [28] S. Sen, L. Gao, J. Rexford, and D. Towsley. Optimal pathcing scheme for efficient multimedia streaming. In *Proc. of NOSSDAV*, June 1999.
- [29] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proceedings of IEEE Infocom '99*, New York, USA., 1999.
- [30] S. Sheu, K. A. Hua, and T. H. Hu. Virtual batching: A new scheduling technique for video-on-demand servers. In *Proc. of the 5th DASFAA*, Melbourne, Australia, Apr 1997.
- [31] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *ACM Multimedia Systems*, pages 197–208, 1996.

Appendix A: Proof of Theorem 1

The arrival of requests for a particular clip is Poisson with rate λ . Therefore, the joint probability distribution of (z_1, z_2, \dots, z_n) is given by:

$$P(m_1, m_2, \dots, m_n) = \prod_{i=1}^n \alpha^{m_i} (1 - \alpha)$$

The probability that there are exactly n patch segments is given by

$$Q_n = \sum_{S_n} P(m_1, m_2, \dots, m_n)$$

$$S_n = \left\{ (m_1, m_2, \dots, m_n) \mid y - nx \leq \sum_i m_i < y - (n-1)x - m_n \right\} = S_n^{(1)} \cup S_n^{(2)}$$

$$\text{where } S_n^{(1)} = \left\{ (m_1, m_2, \dots, m_n) \mid y - nx \leq \sum_i m_i < y - (n-1)x \right\}$$

$$\text{and } S_n^{(2)} = \left\{ (m_1, m_2, \dots, m_n) \mid \sum_{i=1}^{n-1} m_i < y - (n-1)x \leq \sum_i m_i \right\}$$

The conditional probability distribution function, $P(z_i | n = m_i)$, may be rewritten as:

$$P(z_i | n = m_i) = P(z_i | S_n^{(1)} = m_i) P(S_n^{(1)} | n) + P(z_i | S_n^{(2)} = m_i) P(S_n^{(2)} | n)$$

Consider $P(z_i | \mathcal{S}_n^{(1)} = j_i)$, $i = 1, 2, \dots, n$. This is given by:

$$P(z_i = j_i | \mathcal{S}_n^{(1)}) = \frac{\sum_{\{(m_1, m_2, \dots, m_n) \in \mathcal{S}_n^{(1)}, m_i = j_i\}} P(m_1, m_2, \dots, m_n)}{\sum_{\{(m_1, m_2, \dots, m_n) \in \mathcal{S}_n^{(1)}\}} P(m_1, m_2, \dots, m_n)}$$

Observe that $P(m_1, m_2, \dots, m_n)$ and $\mathcal{S}_n^{(1)}$ are symmetric with respect to index i . Hence, we conclude that $P(z_i | \mathcal{S}_n^{(1)} = j_i)$, $i = 1, 2, \dots, n$, are identically distributed. The mean is given by $\frac{1}{n} E_{\mathcal{S}_n^{(1)}}(\sum z_i)$.

But $(\sum z_i | \mathcal{S}_n^{(1)})$ is a random variable taking values between $y - nx$ and $y - (n-1)x + 1$. We approximate this by a uniformly distributed random variable. Hence, we have:

$$EZ_n^{(1)} = E_{\mathcal{S}_n^{(1)}}(z_i) = \frac{1}{n} \left(\frac{y - nx + y - (n-1)x + 1}{2} \right) \approx \frac{y + x/2}{n} - x$$

Now, consider $P(z_i | \mathcal{S}_n^{(2)} = j_i)$, $i = 1, 2, \dots, n-1$. This is given by

$$P(z_i = j_i | \mathcal{S}_n^{(2)}) = \frac{\sum_{\{(m_1, m_2, \dots, m_n) \in \mathcal{S}_n^{(2)}, m_i = j_i\}} P(m_1, m_2, \dots, m_n)}{\sum_{\{(m_1, m_2, \dots, m_n) \in \mathcal{S}_n^{(2)}\}} P(m_1, m_2, \dots, m_n)}$$

Observe that $P(m_1, m_2, \dots, m_n)$ and $\mathcal{S}_n^{(2)}$ are symmetric with respect to index i , for $i = 1, 2, \dots, n-1$. Hence, we conclude that $P(z_i | \mathcal{S}_n^{(2)} = j_i)$, $i = 1, 2, \dots, n-1$, are identically distributed. Their mean is given by $EZ_n^{(2)} = \frac{1}{n-1} E_{\mathcal{S}_n^{(2)}}(\sum_{i=1}^{n-1} z_i)$.

Now, consider $\zeta_n = y - 1 - (n-1)x - \sum_{i=1}^{n-1} z_i$.

$$P(\zeta_n = k) = \frac{\sum_{\{(m_1, m_2, \dots, m_n) | \sum_{i=1}^{n-1} m_i = y - k - 1 - (n-1)x\}} \prod_{i=1}^n \alpha^{m_i} (1 - \alpha)}{\sum_{\{(m_1, m_2, \dots, m_n) \in \mathcal{S}_n^{(2)}\}} \prod_{i=1}^n \alpha^{m_i} (1 - \alpha)} \quad \text{or}$$

$$P(\zeta_n = k) = \frac{\sum_{\{(m_1, m_2, \dots, m_n) | \sum_{i=1}^{n-1} m_i = y - k - 1 - (n-1)x\}} \left(\prod_{i=1}^{n-1} \alpha^{m_i} (1 - \alpha) \right) \alpha^k}{\sum_{\{(m_1, m_2, \dots, m_n) \in \mathcal{S}_n^{(2)}\}} \prod_{i=1}^n \alpha^{m_i} (1 - \alpha)}$$

Compare this with:

$$P(z_1 = k) = \frac{\sum_{\{(m_2, \dots, m_n) | \sum_{i=2}^{n-1} m_i + \zeta_n = y - k - 1 - (n-1)x\}} \prod_{i=2}^n \alpha^{m_i} (1 - \alpha)}{\sum_{\{(m_1, m_2, \dots, m_n) \in \mathcal{S}_n^{(2)}\}} \prod_{i=1}^n \alpha^{m_i} (1 - \alpha)}$$

$$P(z_1 = k) = \frac{\sum_{\{(m_2, \dots, m_n) | \sum_{i=2}^{n-1} m_i + \zeta_n = y - k - 1 - (n-1)x\}} \left(\prod_{i=2}^n \alpha^{m_i} (1 - \alpha) \right) \times \alpha^k}{\sum_{\{(m_1, m_2, \dots, m_n) \in \mathcal{S}_n^{(2)}\}} \prod_{i=1}^n \alpha^{m_i} (1 - \alpha)}$$

Observe that the two marginal distributions are similar in form. Further,

$$\mathcal{S}_n^{(2)} = \left\{ (z_1, z_2, \dots, z_n) \mid \sum_{i=1}^{n-1} z_i + \zeta_n = y - (n-1)x - 1 \quad \text{and} \quad \zeta_n, z_i \geq 0 \right\}$$

which means the set $\mathcal{S}_n^{(2)}$ is symmetric with respect to $z_1, z_2, \dots, z_{n-1}, \zeta_n$. Hence, it is seen that ζ_n has the same distribution as z_i , $i < n$. This gives:

$$EZ_n^{(2)} \times (n-1) + EZ_n^{(2)} = y - (n-1)x - 1 \rightarrow EZ_n^{(2)} = \frac{y + x - 1}{n} - x$$

Thus, we have:

$$EZ|_n = P(\mathcal{S}_n^{(1)})EZ_n^{(1)} + P(\mathcal{S}_n^{(2)})EZ_n^{(2)} = \frac{y + x/2}{n} - x + (x/2n - 1) \times P(\mathcal{S}_n^{(2)})$$

This may be simplified to $EZ|_n \approx \frac{y+x/2}{n} - x$.

Appendix B: Proof of Theorem 2

From Figure 2, it can be seen that each patch segment consists of a geometrically distributed time interval with no requests, followed by a fixed-size interval of length x . Therefore, the size of each patching segment is an i.i.d. random variable whose distribution is given by:

$$b(n) = \begin{cases} (1 - \alpha)\alpha^{n-x} & n \geq x \\ 0 & n < x \end{cases}$$

Taking the Z -transform, we get:

$$B(z) = \sum_n b(n)z^n = z^x \frac{1 - \alpha}{1 - \alpha z}$$

EN is the mean number of patch segments in a patching window. Recall that the last patch segment in a patch window begins at or before the y^{th} time slot. Therefore, there are $j + 1$ or more patch segments in the patching window if the sum of the lengths of the first j patch segments, i.e., PS_1, PS_2, \dots, PS_j is $< y$. Let the probability of this event be $S_j(y)$. Therefore:

$$EN = \sum_{j=0}^{\infty} S_j(y) \quad (9)$$

It is clear that EN is a monotonically increasing function of y , and may be rewritten as $\mathcal{EN}(y)$. Taking the Z -transform on both sides of 9, we get:

$$M(z) = \sum_{u=0}^{\infty} \mathcal{EN}(u)z^u = \sum_{u=0}^{\infty} \sum_{j=0}^{\infty} S_j(u)z^u \quad (10)$$

Let $R_j(u)$ be the probability that the sum of the the lengths of the first j patch segments, i.e., PS_1, PS_2, \dots, PS_j is equal to u . Then, we have:

$$R_j(u) = \sum_{v=0}^u R_{j-1}(v)B(u - v)$$

$$R_j(z) = \sum_{u=0}^{\infty} R_j(u)z^u = [B(z)]^j$$

Also,

$$S_j(y) = \sum_{u=0}^{y-1} R_j(u)$$

$$S_j(z) = \sum_{u=0}^{\infty} S_j(u)z^u = R_j(z) \frac{z}{1 - z} = [B(z)]^j \frac{z}{1 - z}$$

Therefore, (10) reduces to:

$$M(z) = \frac{z}{(1 - z)(1 - zB(z))} = \frac{z(1 - \alpha z)}{(1 - z)(1 - \alpha z - z^{x+1}(1 - \alpha))} \quad (11)$$

As can be seen, $\mathcal{EN}(y)$ turns out to be a non-polynomial function of y . In order to find the optimal patching cut-off parameter y , we need to deal with a simpler function. As noted earlier,

EN increases with y . So, we approximate with a linear function, EN^γ , such that the area under $EN - EN^\gamma$ is equal to zero. Taking the Z -transform, the value of $M^\gamma(z) - M(z)$ is zero at $z = 1$.

Assuming $EN^\gamma = Cy$, we have $M^\gamma(z) = \frac{C}{(1-z)^2}$.

$$M^\gamma(z) - M(z) = \frac{1}{(1-z)^2} \times \left(C - \frac{1-\alpha z}{1+z+z^2+\dots+z^x - \alpha(z+z^2+\dots+z^x)} \right) \xrightarrow{z \rightarrow 1} 0$$

Therefore, we have:

$$C = \frac{1-\alpha}{x(1-\alpha)+1}$$

Appendix C: Proof of Theorem 3

Consider the last patch segment PS_n of a patching window that contains exactly n patch segments. As seen in Appendix A, the set of events resulting in a patching window consisting of exactly n patch segments is a union of two disjoint sub-sets $\mathcal{S}_n^{(1)}$ and $\mathcal{S}_n^{(2)}$. When an event in $\mathcal{S}_n^{(2)}$ occurs, it can be seen that there is no request arriving between the end of the $(n-1)^{st}$ patch segment and the y^{th} slot. Therefore, no patch is transmitted. The length of the n^{th} patch is trivially zero.

Now, consider an event in $\mathcal{S}_n^{(1)}$. The length of the patch to be transmitted is $\leq y + x$. We approximate this by y . Therefore, the mean length of the last patch conditioned on having exactly n patch segments is given by:

$$P_\Omega^{(n)} \approx y \times P(\mathcal{S}_n^{(1)}|n)$$

Unconditioning on n , we get

$$P_\Omega = \sum_n P_\Omega^{(n)} \approx y \times \sum_n P(\mathcal{S}_n^{(1)})$$

Approximating the patch segments with a Stationary Renewal Process, we can show that $\sum_n P(\mathcal{S}_n^{(1)}) \approx \frac{x(1-\alpha)}{x(1-\alpha)+\alpha}$. Thus, we have:

$$P_\Omega \approx y \times \frac{x(1-\alpha)}{x(1-\alpha)+\alpha}$$