Proxy-Assisted Techniques for Delivering Continuous Multimedia Streams

Lixin Gao, Zhi-Li Zhang, and Don Towsley, *Fellow, IEEE*

***Abstract*—We present a proxy-assisted video delivery architecture that can simultaneously reduce the resources requirements at the central server and the service latency experienced by clients (i.e., end users). Under the proposed video delivery architecture, we develop and analyze two novel proxy-assisted video streaming techniques for on-demand delivery of video objects to a large number of clients. By taking advantage of the resources available at the proxy servers, these techniques not only significantly reduce the central server and network resource requirements, but are also capable of providing near-instantaneous service to a large number of clients. We optimize the performance of our video streaming architecture by carefully selecting video delivery techniques for videos of various popularity and intelligently allocating resources between proxy servers and the central server. Through empirical studies, we demonstrate the efficacy of the proposed proxy-assisted video streaming techniques.**

I. Introduction

THE past few years have seen a dramatic growth of multimedia applications which involve video streaming over the Internet. Server and network resources (in particular, server I/O bandwidth and network bandwidth) have proven to be a major limiting factor in the widespread usage of video streaming over the Internet. In order to support a large population of clients, techniques that efficiently utilize server and network resources are essential. In designing such techniques, another important factor that must be taken into consideration is the *service latency*, i.e., the time a client has to wait until the object he/she has requested is started to playback. The effectiveness of a video delivery technique must be evaluated in terms of both the server and network resources required for delivering a video object and the expected service latency experienced by clients. Clearly, the "popularity" or access pattern of video objects (i.e., how frequently a video object is accessed in a given time period) plays an important role in determining the effectiveness of a video delivery technique.

Manuscript received March 7, 2002; revised April 30, 2002; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Ammar. The work of L. Gao was supported in part by the National Science Foundation under Grant ANI-9977555 and NSF CAREER Award Grant ANI-9875513. The work of Z.-L. Zhang was supported in part by the National Science Foundation under NSF CAREER Award Grant NCR-9734428 and NSF Grant ANI-9903228. The work of D. Towsley was supported in part by the National Science Foundation under Grant ANI-9805185.

L. Gao is with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01060 USA (e-mail: lgao@ecs.umass.edu).

Z.-L. Zhang is with the Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: zhzhang@cs.umn.edu).

D. Towsley is with the Department of Computer Science, University of Massachusetts, Amherst, MA 01030 USA (e-mail: towsley@cs.umass.edu).

Digital Object Identifier 10.1109/TNET.2003.820423

In this paper, we propose a proxy-assisted video streaming architecture that takes advantage of the resources (processing and disk storage) available at proxy servers to significantly reduce the server and (backbone wide-area) network resource requirements, while at the same time providing near-instantaneous service to clients. The central server multicasts video objects periodically, using, for example [13], source-specific multicast. Proxy servers are strategically placed between, say, local access networks and the backbone wide-area network. A proxy server stores a fixed number of initial frames or a "prefix" of the multimedia stream [12] so as to serve the future requests: when a new request arrives, the client joins an on-going multicast group to retrieve the multicast stream from the central server and retrieves the missing initial frames from the proxy server. The missing portion of the prefix is delivered by the proxy using a unicast channel and played back immediately by the client. Hence, the proxy server reduces the service latency experienced by the user by unicasting a prefix of the multimedia stream.

This proxy-assisted video delivery environment has several advantages over the traditional stand-alone video server environment. First, since it requires only network resources between the proxy and the client, latency reduction is achieved without increasing the demand on backbone network resources. Second, unlike the proxy caching schemes proposed for conventional Web objects such as text and image objects, the proxy needs to store only prefixes of the multimedia streams. Thus it is feasible even with the large data volume typically associated with multimedia objects. Third, since the proxy server delivers only the prefixes and is only responsible for a limited number of clients, the I/O bandwidth requirement imposed on the proxy server is not significant.

Under the proposed proxy-assisted video delivery architecture, we develop a novel video delivery technique called *proxy-assisted catching*, which can efficiently utilize server and proxy resources while providing near instantaneous service to clients. This technique is particularly suitable for "hot" (i.e., frequently access) video objects. The effectiveness of the technique is achieved through the intelligent integration of the "server-push" and "client-pull" video delivery paradigms. Using this technique, the server periodically "broadcasts" a video object via a number of *dedicated multicast channels*. A client who wishes to watch the video immediately joins an appropriate multicast channel without waiting for the beginning of the next broadcast period. At the same time, the client sends a request to a proxy server to retrieve the missing prefix of the video object. Using a smart broadcast scheme such as the *Greedy Disk-conserving Broadcast* (GDB) scheme [7], we present an analytical framework to determine design

parameters such as the size of the prefix stored in the proxy server. Furthermore, we show that the total resource required by the central server and proxy servers combined is close to the minimum achievable by any broadcasting scheme that supplies near-instantaneous service.

In order to account for the diverse access patterns for a collection of video objects in a video server, we design an efficient video delivery scheme called *proxy-assisted selective catching* which combines proxy-assisted catching with another video delivery technique—*controlled multicast* [8]. Controlled multicast is a "client-pull" technique which is most effective in delivering "cold" video objects. Based on video access patterns, we introduce a simple policy for classifying "hot" and "cold" video objects and apply catching and controlled multicast accordingly to deliver the video objects to clients. Through empirical studies, we demonstrate that, in terms of both network resource requirements and service latency, *proxy-assisted selective catching* outperforms either proxy-assisted catching or controlled multicast applied alone.

The remainder of this paper is organized as follows. The related work is briefly surveyed in Section I-A. Section II presents the proxy-assisted video delivery architecture. In Section III, we describe a specific proxy-assisted video delivery technique called proxy-assisted catching. Section IV introduces proxy-assisted selective catching and evaluates the scheme via empirical studies. The paper is concluded in Section V.

A. Related Work

In recent years, a variety of multicast techniques for video delivery have been proposed (see, e.g., [1], [3], [4], [6], [7], and [18]). These techniques can be broadly classified into either "client-pull" or "server-push." The simplest "client-pull" technique is to deliver a separate video stream upon each client request. This technique, while providing minimal service latency to a client, is obviously not efficient in terms of server and network resource utilization. Clever "client-pull" techniques such as *batching* [1], [6] and *patching* [5], [8], [15], [16] have been proposed that take advantage of the underlying network multicasting capabilities to reduce server and network resource requirements. In the case of batching, this reduction in server and network resource requirements is achieved through increased service latency, as it delays earlier requests for a video object until a certain number of requests for the same object arrive before the video object is scheduled to be delivered. Hence, batching is less effective for "cold" video objects. On the other hand, "patching," which allows multiple clients to share a multicast channel whenever possible, is most effective in reducing the server and network resource requirements for "cold" video objects without introducing service latency. A similar technique—the split and merge (SAM) protocol—is proposed in [14] for *interactive* VOD systems, where a unicast stream is scheduled on demand upon a client's request.

"Server-push" techniques [3], [4], [7], [17]–[19] are typically designed for "hot" video objects. They employ a fixed number of multicast channels to periodically broadcast video objects to a group of subscribers. The difference between various "server-push" techniques lies in the broadcast schemes used. These broadcast schemes determine the server and network resources required for broadcasting a video object. "Server-push" techniques have the advantage that they utilize server and network resources more efficiently. But this efficiency is achieved through increased service latency, as a client can only start receiving a video object at the beginning of next broadcast period.

The problem of delivering continuous media streams using proxy servers has been studied in a number of contexts. In [11], Wang *et al.* develop video staging techniques to store a pre-determined amount of video streams in strategically placed proxy servers to reduce the backbone network bandwidth requirement for delivering variable-bit-rate (VBR) video streams across a wide-area network. In [12], a prefix caching scheme is proposed to reduce the latency while delivering smoothed VBR continuous streams between the proxy and clients. Proxy-assisted video delivery is also proposed in the context of the dynamic skyscraper delivery scheme in [9].

II. Proxy-Assisted Video Delivery Architecture

In this section we propose a proxy-assisted video delivery architecture that employs a central-server-based periodic broadcast scheme to efficiently utilize central server and network resources, while in the same time exploiting proxy servers to significantly reduce service latency experienced by clients. Under the proposed proxy-assisted video delivery architecture, we develop two novel video streaming techniques—*proxy-assisted catching* and *proxy-assisted selective catching*. The proxy-assisted catching technique eliminates the shortcoming associated with periodic-broadcast-based "server push" techniques, namely, the increased service latency, by taking advantage of the resources available at the proxy servers. The proxy-assisted selective catching technique further improves the overall performance by combining proxy-assisted catching and controlled multicast to account for diverse user access patterns. In addition, unlike [9], our video streaming techniques can handle variable size video objects, and is based on formal analysis of multicast scheduling policies. From this analysis, the design parameters can be derived in a straightforward manner. As a result, our solution can be optimized accordingly. In the following we describe the proposed proxy-assisted video delivery architecture and introduce the necessary notation and terminology. The proxy-assisted catching and proxy-assisted selective catching techniques are presented and studied in Sections III and IV, respectively.

Fig. 1 shows a simple example of a proxy-assisted video delivery system. A central video server delivers video streams from a video object repository to a large number of clients across an inter-network (e.g., the Internet). A number of proxy servers are strategically placed between the wide-area backbone network and the local access networks where clients reside. The central server organizes the central server and network resources required to deliver a video stream[1] into a *data channel*. The server uses a multicast channel to deliver a video stream *periodically* to a group of clients (this group of clients is referred to

[1]In this paper we use the term *video stream* to denote a continuous flow or "stream" of video data (belonging to a certain video object) delivered from the server to a client or a group of clients. As will be clear later, a single video object can be partitioned into segments and delivered using multiple video streams via several delivery channels.

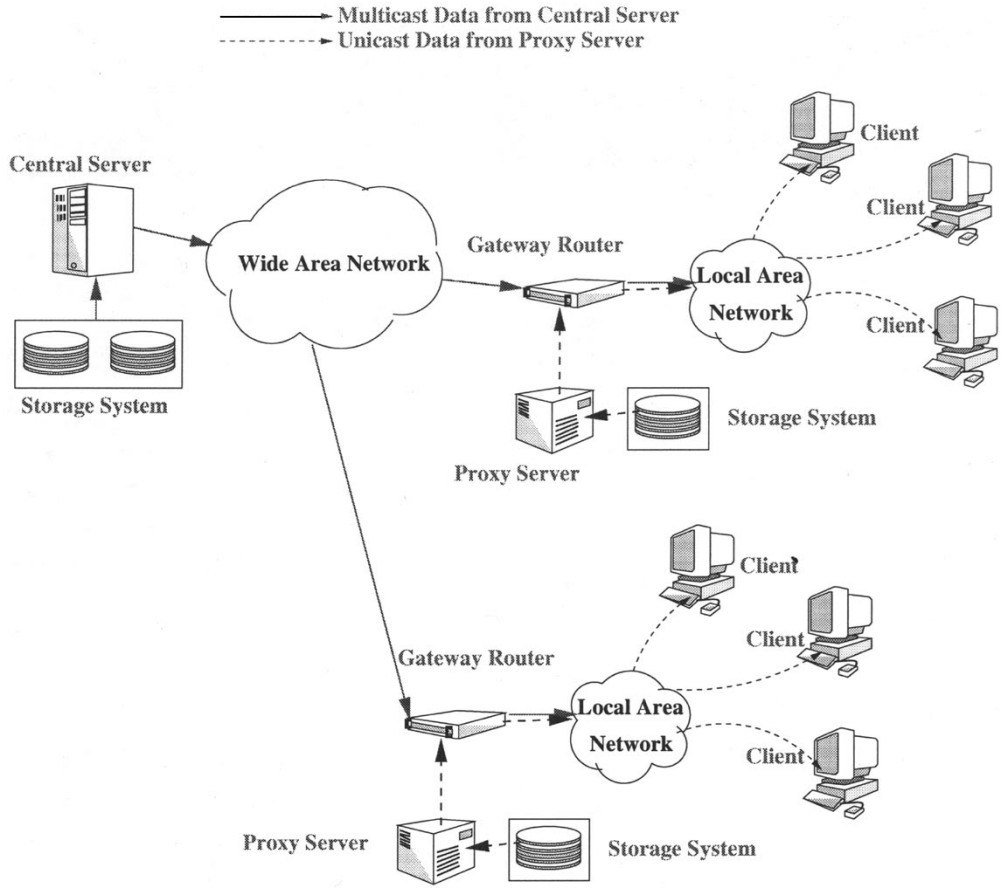

Fig. 1. An overview of proxy-assisted video delivery architecture.

as a *multicast group*). In addition to the logical channels used for delivering video streams (i.e., the data channels), we also assume that there are *control* channels to deliver signaling messages to a client or a group of clients and vice versa for control purposes (e.g., which video object is requested by a client, which data channels a client should tune in to, when to start video play-back, etc.). The video server has a scheduler which receives client requests for video objects via control channels, processes them and determines when and which video delivery channels to deliver requested video objects to clients. Since the bandwidth required by control channels is negligible comparing to that required by data channels, we concern ourselves only with the bandwidth required by the data channels throughout this paper.

The proxy servers *stage* (i.e., pre-store) a fixed number of initial frames of (some) video objects. When a client requests for a video object, it tunes to the central server to fetch its desired video data. However, to ensure near-instantaneous playback, the central server directs client to immediately fetch the initial frames of the video object that is staged at a proxy server that is "closest" to the client.[2] These initial frames are delivered from the proxy server by initiating a unicast channel. By staging a small amount of video data at the proxy, we see that proxy servers can effectively reduce the service latency experienced by the client without increasing the server network bandwidth requirement. In other words, the proxy-assisted video delivery architecture leverages the strategical location of the proxy servers and their storage and processing capacity by appropriately distributing the responsibility of video delivery between the central video server and the proxy servers.

In our work, we assume that each client contains a disk and a video display monitor. A client selects one or more network channels to receive a requested video object according to the instructions from the server. The received video data are either sent to the display monitor for immediate playback, or temporarily stored on the disk which is retrieved and later played back on the display monitor. The *client storage space* is the maximum disk space required throughout the client playback period. For ease of exposition, we assume that the client disk space is sufficiently large to store at least half a video.[3] The *client network bandwidth* is the maximum client network bandwidth required to receive video data from the network throughout the client playback period. We also assume that a client has the ca-

[2]The issue of how to locate the "closest" proxy server is outside the scope of this paper. Such issue has been studied by a number of researchers, e.g., in the context of replicated servers [20]. Other related issues such as proxy server placement, i.e., the number of proxy servers required and where to place them over the Internet are also important to the deployment of the proposed proxy-assisted video delivery architecture; likewise, they are beyond the scope of this paper. Some proposals can be found in, e.g., [21]–[23].

[3]This assumption is not essential, since our proposed schemes can be easily extended to a general case where clients have any given amount of disk storage space, as we will point out in Section III. In all of our empirical studies, the amount of client disk storage space used is actually only at most one third of a video object.

pability of receiving video data from two channels at the same time.[4]

Throughout the paper, we assume that that there are N video objects in the central server object repository. The length of the ith object, $i = 1, 2, \ldots, N$, is L_i minutes long. The requests for the ith video object arrive according to a Poisson distribution with an expected inter-arrival time of $1/\lambda_i$, where λ_i is the request rate of video i. Given a client request for a video object, the *service latency* experienced by a client is the amount of time that the client has to wait until he/she can start the playback of the requested video object. A key issue in the design of proxy-assisted video streaming techniques is how to efficiently utilize server, proxy server and network resources (i.e., use the least number of video delivery channels necessary for delivering a video object) while keeping the (expected) service latency experienced by clients as small as possible. In the remainder of this paper we will illustrate how this issue can be effectively addressed under our proposed proxy-assisted video delivery architecture by introducing two novel video streaming techniques—the proxy-assisted catching and proxy-assisted selective catching techniques.

III. Proxy-Assisted Catching

In this section, we present the proxy-assisted catching technique developed under our proposed video delivery architecture. The basic scheme is described in Section III-A, and its optimality is analyzed in Section III-B. In Section III-Cwe compare the performance of the proxy-assisted catching technique with that of a previously proposed "client-pull" video streaming technique, the *controlled multicast* [8].

A. The Basic Scheme

Although proxy-assisted video delivery techniques can be combined with any video multicast scheme, we illustrate the idea based on a specific periodic broadcasting scheme called Greedy Disk-conserving Broadcast (GDB) [7] (see Appendix I for a detailed description of GDB). We refer to the video delivery technique illustrated here as *proxy-assisted catching*. Proxy-assisted catching achieves the resource efficiency of periodic broadcast schemes while providing near instantaneous service to clients. This technique synergetically combines the "server-push" and "client-pull" video delivery paradigms: like periodic broadcast, proxy-assisted catching dedicates a certain number of server channels for periodic broadcasting. But unlike periodic broadcast, it reduces the service latency by allowing clients to join an on-going broadcast cycle at any time instead of waiting for the next broadcast cycle. Clients catch up with the current broadcast cycle by retrieving the missing initial frames (or a prefix) of the video object from a local proxy server via a separate unicast channel. Clients play back the prefix as it is received from the proxy, while at the same time temporarily storing the video data received from the broadcast channel into the disk.

We illustrate how proxy-assisted catching works through a simple example. As shown in Fig. 2, a video object is partitioned into four segments, A, B, C, and D, where A, B, and C are of equal length and D is twice as long. The server dedicates four channels to broadcast the four segments separately. We assume that a proxy server stores segment A on its disk. Consider client 1 who arrives t seconds after the beginning of the current broadcast cycle of segment A. It joins the ongoing broadcast cycle of segment A to receive the remaining data of segment A. At the same time it initiates a unicast "catch-up channel" over which the proxy delivers the first t seconds of video data of segment A to client 1. This "catch-up" stream is played back immediately by client 1, while the broadcasted data of segment A is temporarily stored. At the end of the current broadcast cycle of segment A, client 1 starts receiving segment B by joining the next broadcast cycle of segment B, while continuing the playback of segment A. At the end of the broadcast cycle of segment B, client 1 starts receiving both segments C and D. By temporarily storing video data that does not need to be played back immediately, client 1 ensures the continuous playback of the video object. The behavior of client 2 is similar. The only difference is that, at the end of the broadcast cycle of segment B, client 2 only needs to join the next broadcast cycle of segment C. As a result, client 2 needs to buffer at most s seconds of video data at any time, whereas client 1 needs to buffer up to $t + d$ seconds of video data. Note that, in both cases, clients 1 and 2 receive video data from at most two channels at any given time.

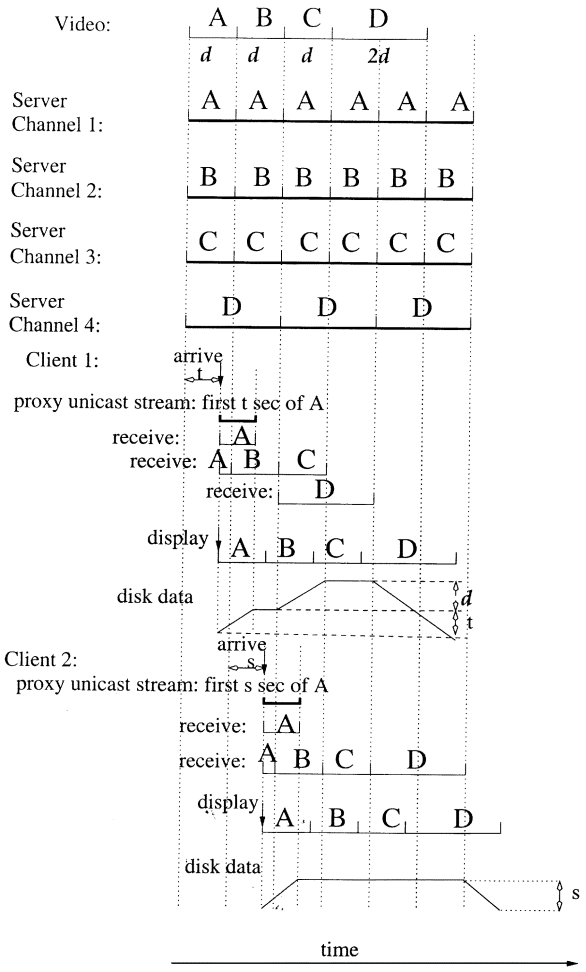

Fig. 2. Illustration of proxy-assisted catching.

[4]With the advent of high-speed access technologies such as ADSL and cable modems, this is not an unreasonable assumption.

From the above example, we see that we can use a function to partition the video into segments. The partition function represents the relative length of each segment. See Appendix I for the partition function used for GDB. The partition function for the proxy-assisted catching is derived directly from that used in GDB. For example, given that the network bandwidth on the client side is only sufficient to support *two* channels, the partition function for catching is given by

$$f(n) = \begin{cases} 1, & \text{if } n \leq 3 \\ 2, & \text{if } n = 4, 5 \\ 5, & \text{if } n = 6, 7 \\ 12, & \text{if } n = 8, 9 \\ 5f(n-4), & \text{if } n > 9. \end{cases} \quad (1)$$

We see that $f(n)$ is derived from f_{GDB} in Appendix I by adding two initial segments whose length is the same as that of the first segment of f_{GDB}. In other words, $f(1) = f(2) = f_{\text{GDB}}(1)$ and $f(n) = f_{\text{GDB}}(n-2)$ for $n \geq 3$. These two segments are added to ensure that a client needs to join only one broadcast channel while receiving the "catch-up" stream from the server. Note that no matter when a client arrives during a broadcast cycle of the first segment, by the end of the broadcast cycle of the second segment, the "catch-up" stream of the first segment must have completely received and played back. Hence, after the end of broadcast cycle of the second segment, a client only needs to join the broadcast channels to receive the appropriate video segments. The client schedule for determining which channels to join and when to join is exactly the same as the one used in GDB. The only difference is that, in proxy-assisted catching, a client always needs to fetch the first t seconds of video data from the proxy server via a unicast channel, if it arrives t seconds later than the beginning of an on-going broadcast cycle of the first segment.

B. Optimality of Proxy-Assisted Catching

Clearly, the ability of proxy-assisted catching to achieve near-instantaneous service lies in the fact that, besides the dedicated broadcast channels, there are additional proxy or central server channels that unicast "catch-up" prefix streams on-demand. Hence the performance of catching is determined by the number of dedicated channels used by the server for periodic broadcast as well as by the number of proxy's "catch-up" channels. The rest of this section is devoted to the analysis of the *optimal* performance of proxy-assisted catching for a video object with a known user access pattern.

Since the server and network bandwidth is the major bottleneck, our goal is to minimize the total number of channels that the server has to dedicate and needed by the proxy server. There is a trade-off however; the fewer channels the server dedicates for periodic broadcast, the longer the first video segment must be and, therefore, the more "catch-up" channels and the more storage space are required by the proxy. Clearly, there is a trade-off between the server and proxy resources. Since the storage space is less expensive comparing to disk or I/O bandwidth, we take the total number of channels required by the server and proxies combined as the optimization criteria. Admittedly, the server and proxy channels have different cost. We argue, however, that our analytical approach presented here can be easily generalized to any cost model. In this paper, we present only the analysis that minimizes the total number of the server and proxy channels.

For simplicity of exposition, we will analyze the performance of proxy-assisted catching based on the partition function $f(n)$ defined in (1). Namely, we assume that the network bandwidth on the client side is only sufficient to support two channels at the same time, and that clients have sufficient disk storage space to buffer at least half of the length of a video object in question. At the end of this section, we will briefly discuss how the analysis can be extended to more general cases.

Consider video object i whose length is L_i. Given the partition function $f(n)$, suppose K_i server channels are dedicated to broadcast video object i and let F_i denote the length of the first broadcast segment of video object i. Then from the definition of the partition function, we have

$$L_i = F_i \sum_{n=1}^{K_i} f(n). \quad (2)$$

Under the assumption that client requests for video object i arrive according to Poisson distribution with an average rate λ_i, the *expected* number of proxy channels needed to deliver "catch-up" streams to clients is

$$\frac{\lambda_i F_i}{2}. \quad (3)$$

This is because the expected length of the "catch-up" streams is $F_i/2$. Hence, the total *expected* number of channels required for delivering video object i to clients is

$$K_i + \frac{F_i \lambda_i}{2}. \quad (4)$$

From (2), we see that the smaller the number of dedicated server channels K_i is, the larger the first broadcast segment F_i becomes. On the other hand, from (3), it is clear that the larger F_i results in more proxy channels are needed to deliver "catch-up" streams to clients. Therefore, there is tradeoff between the number of server channels and the expected number of proxy channels required for catching up. In order to optimize resource efficiency, we minimize the total expected number of channels required to deliver each video object. This leads to the following optimization problem:

$$\begin{aligned} \text{Minimize} \quad & K_i + \frac{F_i \lambda_i}{2} \\ \text{subject to} \quad & L_i = F_i \sum_{n=1}^{K_i} f(n). \end{aligned}$$

Let K_i^* be the solution to the above optimization problem. Namely, K_i^* is the optimal number of server channels such that the total expected number of channels required for delivery of video object i is minimized, i.e.,

$$K_i^* = \arg\min_{K_i} \left(\frac{K_i + F_i \lambda_i}{2} \right). \quad (5)$$

Therefore, the expected number of channels required is

$$K_i^\star + \frac{L_i\lambda_i}{\left(2\sum_{j=1}^{K_i^\star} f(j)\right)}. \tag{6}$$

For the partition function $f(n)$ given in (1), K_i^* can be derived analytically, the detail of which is relegated to Appendix II. Here we provide an order estimate for K_i^*. From (1), it is not too hard to verify that $\sum_{j=1}^{K_i} f(j) = O(5^{K_i/4})$. Substituting this into (6) and taking the first-order derivative, we have

$$K_i^\star = O(\log(\lambda_i L_i)). \tag{7}$$

Furthermore, the total expected number of channels required for video i is

$$O(\log(\lambda_i L_i)) \tag{8}$$

since the expected number of proxy channels required [see (6)]

$$\frac{L_i\lambda_i}{\left(2\sum_{j=1}^{K_i^\star} f(j)\right)} = O(1)$$

with optimal K_i^*.

Although the analysis in this section is carried out based on the assumptions that clients only have sufficient network bandwidth to support two channels and that clients have sufficient disk storage space to store half of a video object, these assumptions are not essential. We can easily extend our analysis to cases where clients can support more than two channels by choosing the appropriate partition functions [7]. Furthermore, since the amount of disk storage space required at clients equals to the size of the largest broadcast segment [7], we can restrict the partition function in such a manner that the largest broadcast segment size is always smaller than the available client disk storage space (please refer to [7] to see how this can be done).

We now proceed to prove that the total number of channels required by proxy-assisted catching is close to the minimum achievable by any broadcast scheme that supplies near-instantaneous service. Formally, for a video of length L and whose request arrival is a Poisson process with arrival rate λ, any broadcast scheme needs at least $\log(\lambda L)$ channels to supply the instantaneous service. Consider the jth frame of the video. First, we prove that we need at least $1/(j+1/\lambda)$ channels for broadcasting frame j.

Let random variable X_j denote the time between two consecutive jth frame multicast. Here we use one frame time as our time unit. We claim that $E[X_j] \le j + 1/\lambda$.

Suppose frame j is broadcast at time t. Any client that arrives between time $t-j$ and t might be able to retrieve the same frame multicast at time t. However, the first client that arrives after time t could not. Let s denote the time that the first client arrives after time t. Frame j has to be broadcast once between time t and time $s+j$ to ensure the continuous playback of the first client arriving after time t. Therefore, $X_j \le s + j - t$. Since the arrival process is Poisson with rate λ, we know that $E[s-t] = 1/\lambda$. Therefore, $E[X_j] \le j + 1/\lambda$.

Since we need to use at least $1/E[X_j]$ channels for the jth frame, we need at least $1/(j+1/\lambda)$ channels for frame j. Summing all frames, we need at least

$$\sum_{j=1}^{L} \frac{1}{E[X_j]} \ge \sum_{j=1}^{L} \frac{1}{j+1/\lambda} \approx \log(\lambda L)$$

channels for the whole video. Comparing with (8), we see that the number of channels required by proxy-assisted catching is within a constant factor from the minimum achievable by any broadcasting scheme. The constant factor depends on the optimal K_i^* in (5), which in turn depends on the broadcasting scheme used.

C. Comparison With Proxy-Assisted Controlled Multicast

In this section, we compare proxy-assisted catching with a previously proposed video delivery technique—*controlled multicast* [8]. Controlled multicast is a "client-pull" technique that allocates channels at the request of clients, and is amenable to deployment in a proxy-assisted video delivery architecture. Here we briefly review the idea of controlled multicast. See [8] for details. Controlled multicast only allows clients to share a multicast channel to receive a video stream *when the later client requests for the same video object arrive within a certain time from the first client request*. Otherwise, a complete video transmission for the video object is scheduled using a new multicast channel. In other words, for each video object i, a threshold T_i is defined to control the frequency at which a complete stream of video i is delivered.

In the proxy-assisted controlled multicast, proxies pre-store the first T_i frames of the video i. When the first request for video i arrives at time t, the central server multicasts a complete video stream of video i. Any subsequent request for video i retrieves the data from the same multicast stream so long as the request arrives within T_i minutes from the starting time of the previous multicast (which is time t in this case). The missing portion of the prefix (which is at most T_i frames of the video i) can be retrieved directly from a local proxy via a unicast channel. Otherwise, the request is served by initiating a new multicast transmission of video i from the server. This process repeats forever. The expected total number of channels required by controlled multicast to deliver video i is

$$\sqrt{2L_i\lambda_i + 1} - 1 \tag{9}$$

as given in [8]. It is clear that, for λ_i reasonably large, the total expected number of channels required by proxy-assisted catching to deliver a video object is considerably less than that required by proxy-assisted controlled multicast. In other words, for "hot" video objects, proxy-assisted catching is much more efficient than proxy-assisted controlled multicast. To verify this observation, let us consider an numerical example. In Fig. 3, we plot the total expected number of channels required to deliver a 90-min-long video object under proxy-assisted catching and proxy-assisted controlled multicast, respectively, as the arrival rate of client requests for the video object varies from 0.1 to 0.9. We can see that proxy-assisted catching requires fewer channels than proxy-assisted controlled multicast when the request rate is greater than 0.4. When the request rate drops below

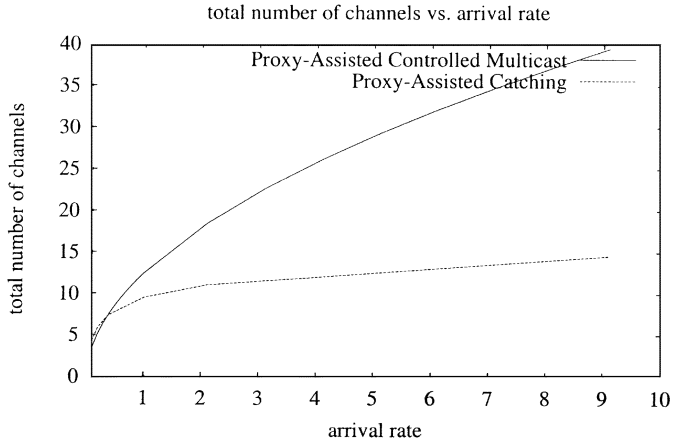

Fig. 3. Total number of channels versus arrival rate.

0.4, proxy-assisted controlled multicast has better performance. This is because for a "cold" video, client requests for the video object do not come frequently enough to take advantage of periodic broadcast.

IV. Proxy-Assisted Selective Catching

In Section III, we developed the proxy-assisted catching video delivery technique and showed that it is most effective for "hot" video objects. For "cold" video objects, proxy-assisted controlled multicast is more efficient. In this section, we design a new video streaming technique, referred to as *proxy-assisted selective catching*, that combines proxy-assisted catching and controlled multicast to account for diverse user access patterns of video objects. The proxy-assisted selective catching broadcasts "hot" videos using the proxy-assisted catching technique, while it delivers "cold" videos using the proxy-assisted controlled multicast technique. In Section IV-A, we provide a formal definition of "hot" and "cold" videos. The rest of this section is devoted to the performance evaluation of the proxy-assisted selective catching technique under various performance metrics.

A. Classification of "Hot" and "Cold" Videos

The key issue in the design of proxy-assisted selective catching is to determine when to apply catching and when to apply controlled-multicast. To address this issue, we present a simple and straightforward criterion for classifying video objects based on their access patterns. A video object is considered *"hot"* if the expected total number of channels required to deliver it using catching is less than that required using controlled multicast. Otherwise, the video object is considered *"cold"*. More specifically, a video object i is considered "hot" if and only if

$$K_i^* + \frac{L_i\lambda_i}{\left(2\sum_{j=1}^{K_i^*} f(j)\right)} > \sqrt{2L_i\lambda_i + 1} - 1 \qquad (10)$$

where K_i^* is defined in (5).

This definition of "hot" and "cold" video objects is based on the assumption that there are always sufficient server channels available (i.e., at least K_i^*) that can be dedicated to broadcast a video object i. The definition can be extended to the case where this is not true by allocating server channels to only "hottest" video objects among the "hot" video objects (say, based on user access patterns). For simplicity of exposition, we will not consider this case.

In the remainder of this section, we conduct simulations to demonstrate that using this simple policy for classifying "hot" and "cold" video objects, the proxy-assisted selective catching technique can achieve superior performance over proxy-assisted catching and proxy-assisted controlled multicast alone. We further show that proxy-assisted selective catching can significantly reduce the server bandwidth requirement.

TABLE I
Parameters Chosen for the Simulation

Simulation Parameters	Default Value
Number of Videos	100
Request Rate (requests/min)	50
Video Length (minutes)	90
Total Number of Channels	700
Disk Size (minutes of data)	30
Skew Factor	0.271

B. Simulation Setting

In our simulations, we assume that client requests arrive at a video server according to a Poisson distribution with an average rate λ (i.e., the average interarrival time between consecutive requests is $1/\lambda$). For a given request, the probability distribution of video selection obeys a Zipf-like distribution [10]: for a collection of N video objects, the probability of selecting video object i, $i = 1,2,\ldots,N$, is $F_i = g_i/\sum_{j=1}^{N} g_j$, where $g_i = 1/i^{1-\theta}$. Here θ denotes the skew factor in video access patterns. In our simulations, we use $\theta = 0.271$. This value of θ is known to closely match the video popularity distributions observed by video rental stores [2].

Unless noted otherwise, the workload and system parameters chosen for our simulations are listed in Table I. Each run of our simulations simulates 150 hours of client requests. In our simulations, the server dedicates a fixed number (K_i^*) of broadcasting channels for each "hot" video objects. The remaining channels are used for broadcasting complete video streams for controlled multicast. These channels are scheduled on an on-demand basis, and requests for these channels are served in First-Come-First-Serve order. In the next two subsections, we demonstrate through an empirical study that the proposed proxy-assisted selective catching can drastically reduce the total number of channels required and the number of channels required on the server alone. Furthermore, the expected service latency of clients can be significantly improved.

C. Total Number of Channels

We first compare the proxy-assisted catching, controlled multicast and selective catching in terms of the total number of channels required. We assume that we have sufficient proxy resources to store prefixes for all videos. Specifically, we assume that each proxy server has 40GBytes of storage space

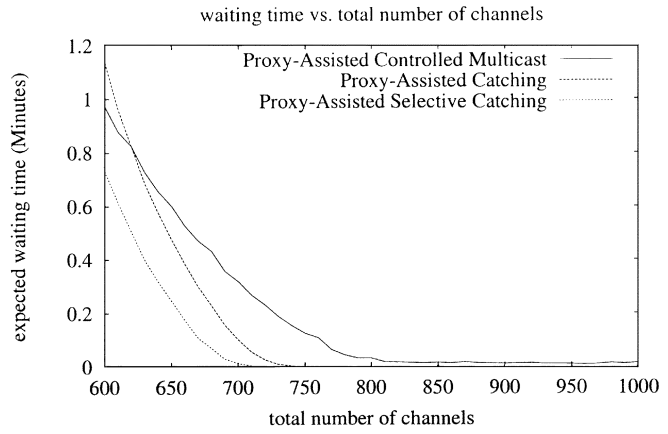

Fig. 4. Expected waiting time versus number of server channels.

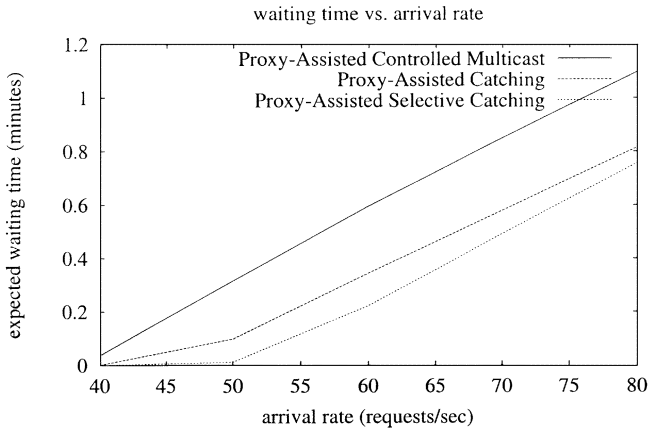

Fig. 5. Expected waiting time vs. arrival rate.

These assumptions are not unreasonable with current trend in disk storage technologies. Fig. 4 compares the expected service latency experienced by clients under proxy-assisted controlled multicast, proxy-assisted catching and proxy-assisted selective catching, as the total number of channels varies from 600 to 1000. The average arrival rate λ of client requests is fixed at 50 requests per minute. From the figure, we see that proxy-assisted catching performs worse than proxy-assisted controlled multicast when the number of total channels is small. This is because proxy-assisted catching dedicates a fixed number of broadcast channels for each video objects no matter whether it is "hot" or "cold". Hence when the total number of channels is small, there are not sufficient channels left for delivering "catch-up" streams. As a result, clients may experience large service latency. But as the number of channels increases, this problem becomes less severe. In these cases, proxy-assisted catching has significantly better performance than proxy-assisted controlled multicast. Proxy-assisted selective catching, however, attains the best performance among the three in all cases. In particular, we observe that with 710 or more channels, proxy-assisted selective catching achieves *zero* expected service latency. To obtain the same result, proxy-assisted controlled multicast requires at least 900 channels.

Fig. 5 shows the effect of the arrival rate on the expected service latency for proxy-assisted controlled multicast, proxy-assisted catching and proxy-assisted selective catching. In this simulation, the request arrival rate varies from 40 to 80 requests per minute, and the total number of channels is fixed at 700. From the figure we see that in most cases, proxy-assisted catching and proxy-assisted selective catching outperform controlled multicast. In particular, as the arrival rate increases from 40 to 80, the benefit of proxy-assisted catching and proxy-assisted selective catching becomes significant. As the arrival rate increases further, we expect that proxy-assisted catching and proxy-assisted selective catching has the same performance since all videos become hot. However, we do not expect the VOD system to perform in the region where the arrival rate is greater than 80 when the number of server channels is 700, since the service latency is unacceptably high and our optimization criteria is based on the assumption that the expected service latency is close to zero. Also note that proxy-assisted selective catching can provide a zero expected service latency even when the request arrival rate is 50 requests per minute, while proxy-assisted catching can provide a zero expected service latency only when the request arrival rate is 40 requests per minute.

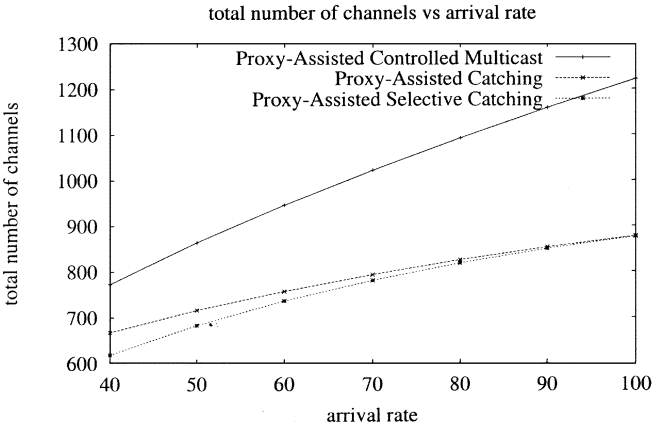

Fig. 6. Expected total number of channels versus arrival rate.

Fig. 6 compares the total expected number of channels required for proxy-assisted controlled multicast, proxy-assisted catching and proxy-assisted selective catching as a function of the request arrival rate. In this simulation, we vary the request arrival rate from 40 to 100 requests per minute. The results show that proxy-assisted selective catching requires at least 150 channels fewer than that required by proxy-assisted controlled multicast, while proxy-assisted catching requires at least 100 fewer channels. As the request arrival rate increases, more video objects become "hot." As a result, the difference between the total number of channels required by proxy-assisted selective catching (or proxy-assisted catching) and proxy-assisted controlled multicast is further widened. In general, proxy-assisted selective catching requires few channels than proxy-assisted catching, and the difference between them diminishes as the request arrival rate increases. As the request arrival rate reaches close to 100, the expected total number of channels required by proxy-assisted catching approaches to that required by proxy-assisted selective catching. This is because as all videos become "hot" proxy-assisted selective catching coincides with proxy-assisted catching.

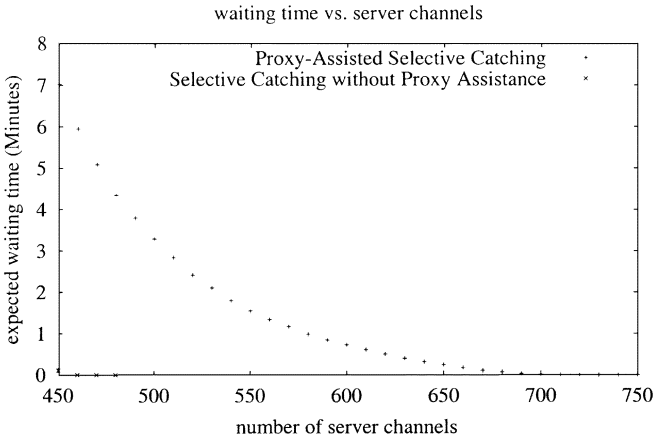

Fig. 7. Expected waiting time versus total number of server channels.

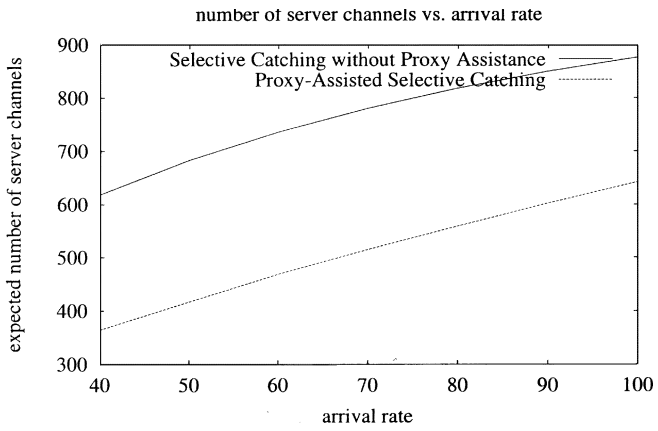

Fig. 8. Expected number of server channels versus arrival rate.

D. Number of Server Channels

We now demonstrate the efficacy of proxy-assisted selective catching scheme in reducing the number of server channels required. We first assume that we have sufficient proxy resources to store prefixes for all videos. Specifically, we assume that each proxy server has 40GBytes of storage space and a disk I/O bandwidth of 88 Mb/s. These assumptions are not unreasonable with current trend in disk storage and LAN access technologies.

Fig. 7 compares the expected service latency of selective catching without proxy assistance and proxy-assisted selective catching, as the number of channels available at the central server increases. We see that that proxy-assisted selective catching can provide an expected service latency close to 0 with only 460 server channels. Whereas, to provide the same service, selective catching without proxy assistance needs 700 channels. We see that proxy-assisted selective catching yields a 36% saving in the number of channels required at the central server. Fig. 8 plots the expected number of channels required at the central server as a function of the request arrival rate. We see that proxy-assisted selective catching achieves significant reduction in the number of central server channel requirement in all the range of the request arrival rates.

In the next set of simulations, we study the resource tradeoff between the central server and the proxy server. We assume that

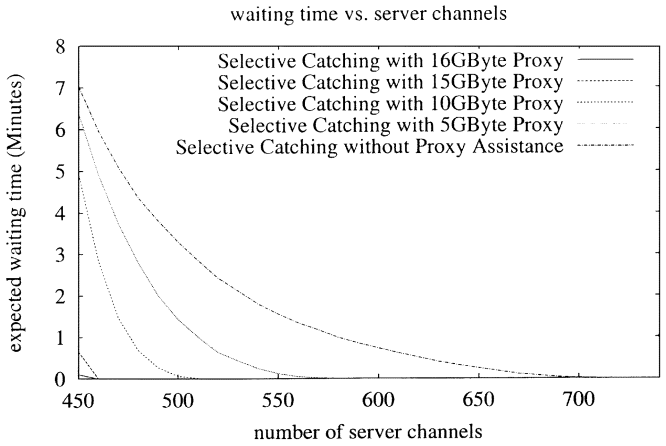

Fig. 9. Expected waiting time versus number of server channels.

the proxy server has limited amount of the storage space. Consequently, we only stage the prefixes of the most popular (i.e., "hottest") videos for a given storage constraint. In Fig. 9 we show the performance of proxy-assisted selective catching as the amount of storage space at the proxy server varies among 5, 10, 15, and 16 GBytes. The expected service latency of proxy-assisted selective catching under these proxy storage constraints as well as that of selective catching without proxy-assistance are plotted. Comparing with the performance of proxy-assisted selective catching, we see that with relative small amount (e.g., 5 GBytes) of proxy storage space, there still is a significant reduction in the expected service latency using proxy-assisted selective catching. Hence, the advantage of proxy-assisted selective catching does not critically hinge on the availability of proxy storage space.

V. Conclusion

In this paper, we have presented a novel and efficient proxy-assisted video delivery architecture that employs a central-server-based periodic broadcast scheme to efficiently utilize central server and network resources, while in the same time exploiting proxy servers to significantly reduce service latency experienced by clients. Under the proposed proxy-assisted video delivery architecture, we have developed two novel video streaming techniques—proxy-assisted catching and proxy-assisted selective catching. By staging at proxy servers the initial streams (i.e., prefixes) of a certain number of video objects, the proxy-assisted catching technique retains the resource efficiency of the periodic-broadcast-based "server-push" schemes, while in the same providing near-instantaneous service to a large number of clients. This technique is particularly effective for "hot" (i.e., frequently accessed) video objects. The proxy-assisted selective catching technique combines the proxy-assisted catching technique for delivery of "hot" video objects with the proxy-assisted controlled multicast technique for delivery "cold" video objects to account for the diverse video access patterns. We presented a simple criterion for classifying video objects into "hot" and "cold" sets. Through simulations we demonstrated that the proposed proxy-assisted selective catching can achieve superior performance over

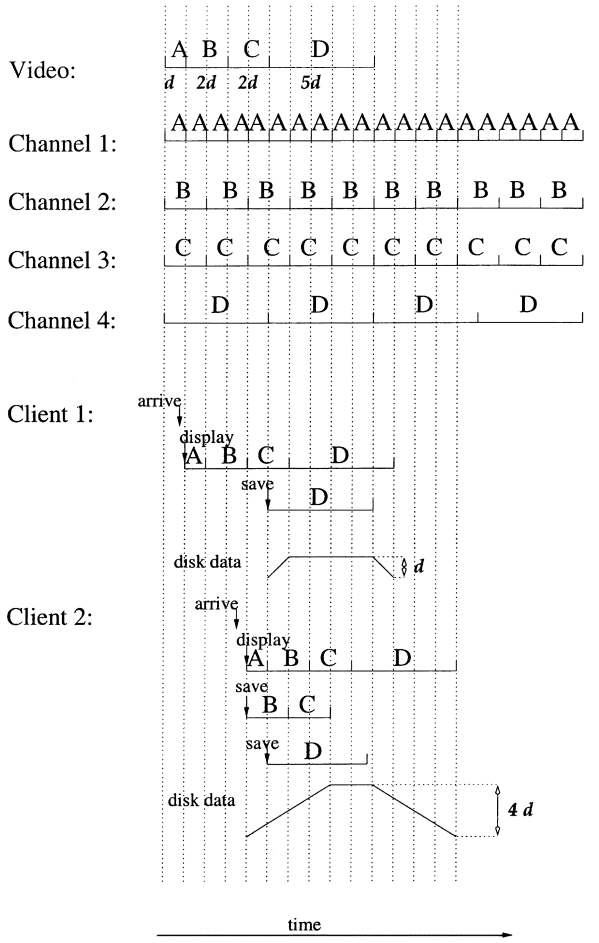

Fig. 10. Example of periodic broadcast scheme.

existing techniques in terms of both server/network resource requirements and service latency.

Appendix I
GDB: An Efficient Periodic Broadcast

Greedy Disk-conserving Broadcast (GDB) [7] is a "server-push" video delivery technique based on the innovative periodic broadcast schemes developed recently [1], [3], [4]. Under these schemes, a video object is partitioned into segments and each segment is periodically broadcasted via a dedicated channel. The sizes of these segments are carefully designed in such a manner that clients who wish to receive the video object can join the appropriate channels to receive various segments at *scheduled* times to ensure continuous playback of the video object. Fig. 10 illustrates how the schemes work through a simple example. A video is partitioned into four segments: A, B, C, D. Each segment is broadcast periodically via a dedicated channel. Clients prefetch video data according to a schedule that ensures the continuous playback. Furthermore, clients are guaranteed a maximum service latency of d minutes with only four dedicated channels, where periodically broadcasting the video stream every d minutes would require 11 dedicated channels to guarantee the same maximum service latency. In effect, a complete stream is multicast every d minutes but using only four channels. The key issue in periodic broadcast schemes is to determine how to partition video objects into segments so as to enable continuous playback at clients. A method for partitioning video objects is referred to as a *partition function*, which determines the performance of a periodic broadcast scheme. Given K_i dedicated multicast channels, a partition function $f(n)$ divides a video object into K_i segments, $T_1, T_2, \ldots, T_{K_i}$, as follows. For $n = 1, 2, \ldots, K_i$, segment T_n contains $f(n)L_i/\sum_{m=1}^{K_i} f(m)$ minutes of video data, and its data starts at the $\sum_{m=1}^{n-1} f(m)L_i/\sum_{m=1}^{K_i} f(m)$th minute of the video and ends at the $\sum_{m=1}^{n} f(m)L_i/\sum_{m=1}^{K_i} f(m)$ th minute of the video. In [7], a set of constraints on partition functions are derived based on resource availability at the client side. In particular, GDB is shown to be most efficient among all the existing schemes, given the same client resource constraints. For example, if the network bandwidth at the client side is only sufficient to support two channels (i.e., receiving from two channels simultaneously), the optimal partition function $f(n)$ used in GDB has the following form:

$$f_{\mathrm{GDB}}(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2, & \text{if } n = 2, 3 \\ 5, & \text{if } n = 4, 5 \\ 12, & \text{if } n = 6, 7 \\ 5f(n-4), & \text{if } n > 7. \end{cases} \tag{11}$$

Given this partition function, it can be shown that the disk storage requirement at the client is equal to $(f_{\mathrm{GDB}}(K_i) - 1)$ times the first segment size for a video object.

Appendix II
Determining the Optimal Number of Dedicated Channels

In this Appendix, we describe how to determine the optimal number of dedicated broadcast channels, i.e., K^* defined in (5). To proceed, we define $h(n) = \sum_{i=1}^{n} f(n)$. It is not hard to verify that

$$h(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2, & \text{if } n = 2 \\ 3, & \text{if} n = 3 \\ 5, & \text{if } n = 4 \\ 7, & \text{if } n = 5 \\ 5^{(n-6)/4} * \frac{27}{2-1.5}, & \text{if } n > 5, n \bmod 4 = 2 \\ 5^{(n-7)/4} * \frac{37}{2-1.5}, & \text{if } n > 5, n \bmod 4 = 3 \\ 5^{(n-8)/4} * \frac{61}{2-1.5}, & \text{if } n > 5, n \bmod 4 = 0 \\ 5^{(n-9)/4} * \frac{85}{2-1.5}, & \text{if } n > 5, n \bmod 4 = 1. \end{cases}$$

Note that, for $j = 0, 1, 2, 3$, $1/h(4k + j + 2)$ is convex in k, $k \geq 0$. Therefore, there exists a $k \geq 0$ such that $2 + (4k + j) + L_i/h(2 + (4k + j))$ is minimized. Let k_j^* denote this value of k. Then $K_i^* \in 1, 2, 3, 4, 5, 4k_0^* + 2, 4k_1^* + 3, 4k_2^* + 4, 4k_3^* + 5$. Determining K_i^* is thus quite straightforward since k_j^*, $j = 0, 1, 2, 3$, can be easily derived from the convexity of $1/h(4k + j + 2)$.

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. IEEE Int. Conf. Multimedia Systems*, June 1996, pp. 118–126.

[2] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On optimal batching policies for video-on-demand storage server," in *Proc. IEEE Int. Conf. Multimedia Systems*, June 1996, pp. 253–258.
[3] S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting," *IEEE Multimedia Syst.*, vol. 4, pp. 197–208, 1996.
[4] K. A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," in *Proc. ACM SIGCOMM*, Sept. 1997, pp. 89–100.
[5] K. A. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true video-on-demand services," in *Proc. ACM Multimedia*, Sept. 1998, pp. 191–200.
[6] A. Dan, P. Shahabuddin, and D. Sitaram, "Scheduling policies for an on-demand video server with batching," in *Proc. ACM Multimedia*, Oct. 1994, pp. 168–179.
[7] L. Gao, J. Kurose, and D. Towsley, "Efficient schemes for broadcasting popular videos," in *Proc. NOSSDAV*, Cambridge, U.K., July 1998.
[8] L. Gao and D. Towsley, "Supplying instantaneous video-on-demand services using controlled multicast," in *Proc. IEEE Multimedia Computing and Systems*, vol. 2, Florence, Italy, June 1999, pp. 117–121.
[9] D. L. Eager, M. C. Ferris, and M. K. Vernon, "Optimized regional caching for on-demand data delivery," in *Proc. MMCN*, Jan. 1999, pp. 301–316.
[10] G. Zipf, *Human Behavior and the Principle of Least Effort*. Reading, MA: Addison-Wesley, 1949.
[11] Y. Wang, Z.-L Zhang, D. H. C. Du, and D. Su, "A network-conscious approach to end-to-end video delivery over wide area networks using proxy servers," in *Proc. IEEE INFOCOM*, 1998, pp. 660–667.
[12] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proc. IEEE INFOCOM*, 1999, pp. 1310–1319.
[13] H. W. Holbrook and D. R. Cheriton, "IP multicast channels: EXPRESS support for large-scale single-source applications," in *Proc. SIGCOMM*, 1999, pp. 65–78.
[14] W. Liao and V. O. K. Li, "The split and merge (SAM) protocol for interactive video-on-demand systems," in *Proc. IEEE INFOCOM*, 1997, p. 1351.
[15] S. W. Carter and D. D. E. Long, "Improving video-on-demand server efficiency through stream tapping," in *Proc. Int. Conf. Computer Communication and Networks (ICCCN'97)*, Las Vegas, NV, Sept. 1997, pp. 200–207.
[16] S. Sen, L. Gao, J. Rexford, and D. Towsley, "Optimal patching schemes for efficient multimedia streaming," in *Proc. NOSSDAV*, June 1999, pp. 265–277.
[17] R. Mondri, "Tailored transmissions for efficient near-video-on-demand service," in *Proc. IEEE Multimedia Computing Systems*, vol. 1, 1999, pp. 226–231.
[18] J.-F. Paris, S. W. Carter, and D. E. Long, "A low bandwidth broadcasting protocol for video on demand," in *Proc. Int. Conf. Computer Communication and Networks (ICCCN)*, Lafayette, IN, Oct. 1998, pp. 690–697.
[19] J.-F. Paris, S. W. Carter, and D. E. Long, "Efficient broadcasting protocols for video on demand," in *Proc. Int. Symp. Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, July 1998, pp. 127–132.
[20] Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar, "A novel server selection technique for improving the response time of a replicated service," in *Proc. IEEE INFOCOM*, San Francisco, CA, Apr. 1998, pp. 783–791.
[21] L. Qui, V. N. Padmanabhan, and G. M. Voelker, "On the placement of web server replicas," in *Proc. IEEE INFOCOM*, Anchorage, AK, Apr. 2001, pp. 1587–1596.
[22] H. S. Bassali, K. M. Kamath, R. B. Hosamani, and L. Gao, "Topology-aware algorithms for proxy placement in the Internet," *J. Comput. Commun.*, vol. 26, no. 3, Feb. 2003.
[23] B. Li, M. Golin, G. Italiano, X. Deng, and K. Sohraby, "On the optimal placement of web proxies in the Internet," in *Proc. IEEE INFOCOM*, vol. 3, New York, Mar. 1999, pp. 1282–1290.

Lixin Gao (M'96) received the Ph.D. degree in computer science from the University of Massachusetts, Amherst, in 1996.

She is an Associate Professor of Electrical and Computer Engineering at the University of Massachusetts. Her research interests include multimedia networking, Internet routing, and security. She was a Visiting Researcher at AT&T Research Labs and DIMACS from May 1999 to January 2000.

Dr. Gao is a principal investigator of a number of the National Science Foundation grants, including an NSF CAREER award. She has been a member of the ACM since 1996, and she is an Alfred Sloan fellow.

Zhi-Li Zhang received the B.S. degree in computer science from Nanjing University, China, in 1986 and the M.S. and Ph.D. degrees in computer science from the University of Massachusetts, Amherst, in 1992 and 1997, respectively.

In 1997, he joined the Computer Science and Engineering faculty at the University of Minnesota, Minneapolis, where he is currently an Associate Professor. From 1987 to 1990, he conducted research in the Computer Science Department at the University, Denmark, under a fellowship from the Chinese National Committee for Education. He has held visiting positions at Sprint Advanced Technology Labs, IBM T.J. Watson Research Center, Fujitsu Labs of America, Microsoft Research China, and INRIA, Sophia Antipolis, France. His research interests include computer communication and networks, especially the QoS guarantee issues in high-speed networks, multimedia and real-time systems, and modeling and performance evaluation of computer and communication systems.

Dr. Zhang currently serves on the Editorial board of IEEE/ACM TRANSACTIONS ON NETWORKING and *Computer Network*. He was a Program Co-chair of the SPIE ITCOM 2002 conference on Scalability and Traffic Control in IP Networks, served on the Executive Committee for IEEE INFOCOM 2001 and INFOCOM 2003, and on the Technical Program Committees of various conferences and workshops including IEEE INFOCOM and ACM SIGMETRICS. He received the National Science Foundation CAREER Award in 1997. He was also awarded the prestigious McKnight Land-Grant Professorship at the University of Minnesota. He is co-recipient of an ACM SIGMETRICS best paper award and an IEEE ICNP best paper award. He is a member of INFORMS Telecommunication Section.

Don Towsley (M'78–SM'93–F'95) received the B.A. degree in physics and the Ph.D. degree in computer science from the University of Texas, Austin, in 1971 and 1975, respectively.

From 1976 to 1985, he was a Member of the Faculty of the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, where he is currently a Distinguished Professor in the Department of Computer Science. He has held visiting positions at IBM T.J. Watson Research Center, Yorktown Heights, NY (1982–1983), Laboratoire MASI, Paris, France (1989–1990), INRIA, Sophia Antipolis, France (1996), and AT&T Labs–Research, Florham Park, NJ (1997). His research interests include networks, multimedia systems, and performance evaluation. He currently serves on the editorial boards of *Performance Evaluation* and *Journal of the ACM*.

Dr. Towsley has served on the editorial boards of the IEEE TRANSACTIONS ON COMMUNICATIONS and the IEEE/ACM TRANSACTIONS ON NETWORKING. He received the 1998 IEEE Communications Society William Bennet Paper Award and three Best Conference Paper Awards from ACM SIGMETRICS. He was a Program Co-Chair of the joint ACM SIGMETRICS and PERFORMANCE'92 Conference. He is a Member of ORSA and Chair of IFIP Working Group 7.3. He is a Fellow of the ACM.