

A Decentralized Prefetching Protocol for VBR Video on Demand *

Martin Reisslein¹, Keith W. Ross², and Vincent Verillotte³

¹ Department of Systems Engineering, University of Pennsylvania,
Philadelphia, PA 19104, USA,
reisslei@seas.upenn.edu,

WWW home page: <http://seas.upenn.edu/~reisslei>

² Institute Eurecom, Sophia-Antipolis 06904, France,
ross@eurecom.fr

WWW home page: <http://www.eurecom.fr/~ross>

³ Sun Microsystems France S.A.,
13 Avenue Morane Saulnier - BP 53, 78142 Velizy Cedex, France,
vverillo@france.sun.com

Abstract. We present a high-performance *decentralized* prefetching protocol for the delivery of VBR video on demand (VoD) from servers to clients across a packet-switched network. The protocol gives constant perceptual quality for high link utilizations. It also allows for immediate commencement of the video upon user request and near instantaneous response to viewer interactions such as pause, resume and temporal jumps. The protocol requires that (1) the client has a moderate amount of memory dedicated to the VoD application (2) the client sends a positive acknowledgment back to the server for each received video frame. Our decentralized prefetching protocol employs window flow control. A send window limits the number of frames a server is allowed to send in a frame period. The send window grows larger than one when the network is underutilized, allowing the server to prefetch future frames into the client memory. When the network becomes congested the send window is reduced and the server is throttled. Simulation results based on MPEG encoded traces show that our decentralized prefetching protocol compares favorably with other prefetching protocols in the existing literature.

1 Introduction

We present a high-performance *decentralized* prefetching protocol for the delivery of video on demand (VoD) from servers to clients across a packet-switched network. The protocol assumes that the videos are variable-bit-rate (VBR) encoded. Not only does this protocol give constant perceptual quality for high link utilizations, but it also allows for immediate commencement of the video upon user request and near instantaneous response to viewer interactions such as pause, resume and temporal jumps.

* Supported partially by NSF grant NCR96-12781

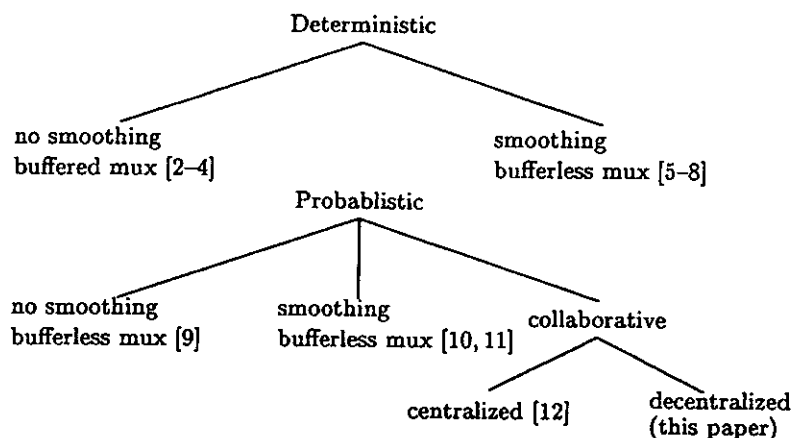


Fig. 1. Transmission Schemes for VBR Video on Demand

To achieve this high performance our protocol has two requirements. First, we require that each client has a moderate amount of memory dedicated to the VoD application. Second, we require that each client sends a positive acknowledgement back to its server for each received video frame. The client could be a television with a set-top box capable of performing buffering and decoding, or it could be a household PC.

Our protocol explicitly assumes that the videos are VBR encoded with high peak-to-mean ratios. The motivation for our approach is that, for the same perceived video quality, Constant Bit Rate (CBR) encoding produces an output rate significantly higher than the average rate of the corresponding VBR encoding for action movies [1]. CBR traffic allows for nearly 100% link utilization; the number of connections that can be carried over a link of given capacity is roughly the link capacity divided by the CBR rate (assuming homogeneous connections). The number of VBR connections that can be transmitted simultaneously is the achievable link utilization multiplied by the link capacity divided by the average rate of the VBR video stream. Therefore schemes for transmitting VBR encoded video that achieve high average link utilizations while keeping losses at a negligible level, can allow for significantly more video connections than does CBR video.

The traffic management schemes for VBR video in the literature fall into four main categories: deterministic; deterministic with smoothing and/or prefetching; probabilistic; and probabilistic with collaborative prefetching; see Figure 1. The deterministic schemes send into the network the original VBR traffic, and admission control ensures that the delays never exceed a prespecified limit [2][3][4]. For highly variable VBR traffic, these deterministic schemes typically require large initial delays to achieve moderate link utilizations [13]. The deterministic schemes with prefetching and smoothing do not send the original VBR traffic

into the network, but instead send some smoothed version of it. Several independent research teams have proposed schemes whereby the server transmits the video at different constant rates over different intervals; these schemes vary in how the rates and intervals are chosen [5][6][7][8][10]. None of the deterministic schemes (with or without prefetching and smoothing) allows for both high link utilizations (>90%) and consistently high responsiveness (less than a second) to interactivity.

For the probabilistic approaches, [9] considers sending the original VBR encoded video into an unbuffered multiplexer. This scheme allows for responsive interactivity, but introduces packet loss whenever the aggregate transmission rate exceeds the link rate. In [10] and [11] related ideas are explored whereby the original traffic is first smoothed before it is statistically multiplexed at an unbuffered link; the statistical multiplexing of the smoothed traffic can substantially increase link utilization at the expense of small packet loss probabilities. In particular, in [11] the authors demonstrate that their prefetching scheme, *Optimal Smoothing*, can give moderately high link utilizations when it is combined with statistical multiplexing.

A probabilistic transmission scheme with collaborative prefetching, *Join-the-Shortest-Queue (JSQ) prefetching*, is presented in [12]. It is shown in [12] that JSQ prefetching has substantially less packet loss than does Optimal Smoothing for the same link utilization. JSQ prefetching achieves nearly 100% link utilization, immediate commencement of playback and instantaneous response to viewer interactions. JSQ prefetching, however, can only be applied when one centralized server feeds many clients. In this paper we introduce a decentralized and collaborative prefetching protocol that allows the video streams to emanate from multiple distributed and decentralized servers.

Our decentralized prefetching protocol performs almost as well as JSQ prefetching: it allows for nearly 100% link utilization, immediate commencement of playback and instantaneous response to viewer interactions.

Our decentralized prefetching protocol employs window flow control; it is inspired by the Transmission Control Protocol (TCP) [14, 15] widely used in the Internet. For simplicity, assume that each server is responsible for exactly one connection. Admission control ensures that all link utilizations do not exceed 95%. Our basic decentralized prefetching protocol works roughly as follows. The server maintains a send window, limiting the number of frames the server is allowed to send in a frame period. The send window is increased by a small increment when all acknowledgments arrive in time. Due to admission control and the VBR nature of the traffic, there are periods of time during which the network is underutilized. The send window grows larger than one during these periods, allowing the server to prefetch future frames into the client memory. In times of network congestion, frames are lost or delayed and the corresponding acknowledgments do not arrive at the server before their timeouts. In this case, the send window is reduced to throttle the server and alleviate the congestion. The reservoir of prefetched frames in the client buffer allows the client to continue playback during these periods of congestion. Starvation at the client occurs only

if the reserve of prefetched frames at the client is completely depleted and the current frame is lost or delayed due to network congestion. We simulate our protocol in the context of a simple network (see Figure 2). The simulations are driven by frame size traces of MPEG 1 encoded videos from the public domain [16]. *Our empirical work indicates that starvation at the client rarely occurs for average link utilizations around 95% and small client buffers.*

This paper is organized as follows. In the following subsection we briefly review two important on demand delivery schemes for VBR-encoded video. In Section 2 we describe our VoD architecture. In Section 3 we introduce our decentralized prefetching protocol. In Section 4 we introduce a number of refinements of the decentralized prefetching protocol. In Section 5 we present simulation results for our decentralized prefetching protocol.

1.1 Review of Transmission Schemes for VBR Video on Demand

In this subsection we review two prefetching schemes for VBR video on demand, Join-the-Shortest-Queue (JSQ) Prefetching [12] and Optimal Smoothing [17, 11, 18]. These two schemes will be used as benchmarks when evaluating our decentralized prefetching protocol.

The JSQ prefetching protocol is suited for the efficient transmission of VBR encoded videos from a video server to a large number of clients with moderate memory. The protocol allows for at most one shared link between the video server and the clients. The policy is based on the observation that due to the VBR nature of the multiplexed traffic there are frequent periods of time during which the shared link's bandwidth is under utilized. During these periods the server prefetches frames from any of the ongoing connections and sends the prefetched frames to the buffers in the appropriate clients. The JSQ policy specifies how the server selects the prefetched frames. The server always selects the next frame from the connection that has the smallest number of prefetched frames in its client's buffer. The JSQ prefetching protocol thus determines the transmission schedule of a connection on-line, as a function of the buffer contents at *all* of the clients. For this reason, JSQ is referred to as a *collaborative prefetching scheme*.

Optimal Smoothing can be applied when transmitting stored video from a server to a client with buffering capabilities across a network. Given a specific client buffer size, the optimal smoothing algorithm determines a "smooth" rate transmission schedule that ensures that the client buffer neither overflows nor underflows. The algorithm is optimal in that it achieves the greatest possible reduction in rate variability. Optimal smoothing is non-collaborative; the transmission schedule is computed before transmission begins and thus does not take the other ongoing connections into account. Admission control for the optimally smoothed trace can be based on the peak-rate of the smoothed trace; this ensures lossless transmission. Another approach is to statistically multiplex the optimally smoothed traces at an unbuffered link and base admission control on a large deviation estimate of the loss probability [9, 11]. We apply the latter approach when comparing optimal smoothing with our decentralized prefetching protocol.

2 Architecture Description

Figure 2 illustrates our basic model for VoD¹. The video servers contain videos

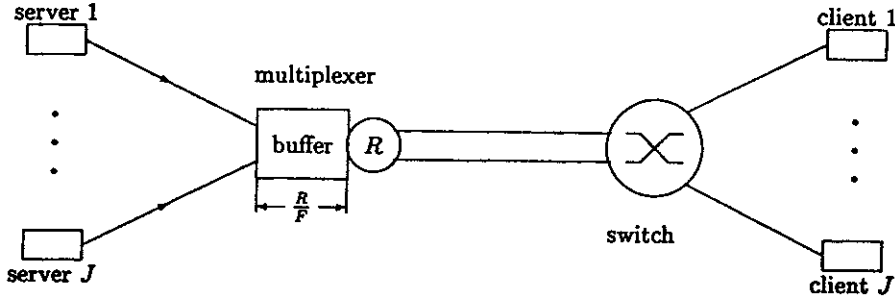


Fig. 2. Video on Demand Architecture

in mass storage. For notational simplicity, assume that each video consists of N frames and has a frame rate of F frames/sec. The videos are VBR encoded using MPEG 1, MPEG 2 or some other video compression algorithm. Let J denote the number of video connections in progress. We assume for the purpose of this study that each video server feeds one client; thus there are J video servers feeding J clients. In explaining the client-server interaction we focus on a particular client-server pair. For simplicity, we assume for the following discussion that each video frame is transmitted in one packet². Let x_n denote the number of bits in the n th frame. Because the videos are prerecorded, the sequence (x_1, x_2, \dots, x_N) is fully known before the transmission of the video. At the beginning of each frame period, that is, every $1/F$ seconds, the server decides according to a prefetching policy, outlined in the next section, which and how many frames to transmit. The server sends the frames to the multiplexer buffer. Frames that do not fit into the multiplexer buffer are lost. The multiplexer buffer of size R/F bit is served at rate R bps. The maximal delay incurred in the multiplexer is therefore $1/F$ seconds. For simplicity we assume that the propagation and processing delays are negligible. The client instantaneously sends a positive acknowledgment to the server for each frame received.

With these delay assumptions, the server receives acknowledgments for all frames successfully received by the client within one frame period. The server therefore knows whether the frames sent in the previous frame period were received before deciding which frames to send in the current frame period.

¹ Although we discuss our protocol in the context of a single shared link, the protocol applies to arbitrary networks with multiple shared links.

² In our numerical work we assume the more realistic case of fixed size packets.

When a client requests a specific video, the network makes an admission control decision by deciding whether or not to grant the request. The admission control policy is to accept connections as long as the average link utilizations are $\leq 95\%$. The average link utilization is $util = F \sum_{j=1}^J x_{avg}(j)/R$, where $x_{avg}(j)$ is the average frame size in bits of the j th connection, which is calculated by averaging the corresponding sequence (x_1, \dots, x_N) . If the network grants the request, a connection is established and the server immediately begins to transmit the connection's frames into the network. The frames arriving at the client are placed in the client's prefetch buffer. The video is displayed on the user's monitor as soon as a few frames have arrived at the client.

Under normal circumstances, every $1/F$ seconds the client removes a frame from its buffer, decompresses it, and displays it. If at one of these epochs there are no complete frames in its prefetch buffer, the client loses the current frame; the client will try to conceal the loss by, for instance, redisplaying the previous frame. At the subsequent epoch the client will attempt to display the next frame of the video.

3 Decentralized Prefetching Protocol

In this section we present our basic decentralized prefetching protocol that allows the server to determine how many frames to send in each frame period. This protocol strives to (1) make efficient use of the buffers at the client and (2) avoid bandwidth "hogging" by a particular connection and thus give each connection a fair share of the bandwidth. The protocol attempts to allow each client to build up a reservoir of prefetched frames. Although our design allows for pause and temporal jumps, we will initially exclude these interactive features. We will also initially assume that the client buffers are infinite.

When discussing the server policy we again focus on a particular connection. We divide time into slots of length $1/F$. Let l denote the current slot; l is a local variable maintained by the server. In the course of the transmission of a video with N frames, l runs from 1 through N . *We do not assume any synchronization of time slots among the client-server pairs.*

Of central importance to our policy is the *send window*, denoted w_l , which limits the amount of traffic the connection can transmit in slot l . Specifically, the server is allowed to transmit $\lfloor w_l \rfloor$ frames during slot l . (We assume for simplicity that only complete frames are transmitted.) A new connection starts with a send window of $w_0 = 1$. The send window is increased by a small increment Δw , say 0.1, at the beginning of each slot, i.e. $w_l = w_{l-1} + \Delta w$. After computing the send window the server transmits $\lfloor w_l \rfloor$ frames; see Figure 3. Note that $w \geq 2$ allows for prefetching of future frames. To keep track of the number of prefetched frames in the client buffer, let p_l be the number of frames in the client buffer at the beginning of slot l . This variable is initialized to $p_1 = 0$. Let a_l denote the number of frames that are received and acknowledged by the client during slot l . Clearly, $0 \leq a_l \leq \lfloor w_l \rfloor$; a_l is equal to $\lfloor w_l \rfloor$ if all frames sent are received by the client. If frames are lost we have $a_l < \lfloor w_l \rfloor$. Figure 3 illustrates the timing of the

prefetching protocol. We assume throughout that multiplexer buffer overflow is

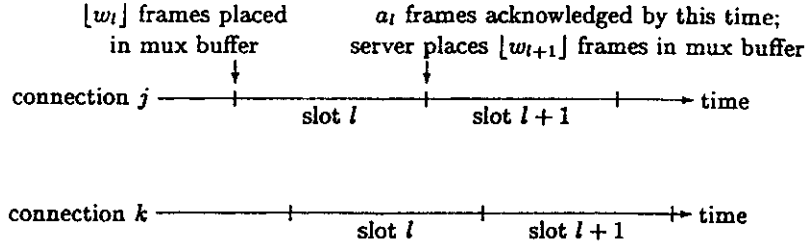


Fig. 3. Timing diagram of prefetching policy. Server j places $\lfloor w_l \rfloor$ frames in the multiplexer buffer at the beginning of slot l . The acknowledgements for a_l frames arrive from the client by the end of slot l . The server processes the acknowledgments and puts $\lfloor w_{l+1} \rfloor$ frames in the multiplexer buffer at the beginning of slot $l+1$. There is no synchronization of slots between any distinct servers j and k

the only source of loss; the switch and interconnecting links are assumed lossless. We also assume that acknowledgements are never lost. Frame l is removed from the client buffer at the end of slot l if the client buffer contains one or more frames. The server keeps track of p_l through the following recursion:

$$p_{l+1} = [p_l + a_l - 1]^+. \quad (1)$$

Let s_l denote the number of bits received and acknowledged by the client during slot l . Let b_l be the number of bits in the client buffer at the beginning of slot l ; initially, $b_1 = 0$. With the given definitions, the server keeps track of b_l through the following recursion:

$$b_{l+1} = [b_l + s_l - x_l]^+. \quad (2)$$

If the server does not receive a positive acknowledgement for a frame sent at the beginning of the previous slot within one frame period, it assumes that the frame is lost. If a connection without any prefetched frames in the client buffer ($p_l = 0$) suffers loss, the client experiences starvation and may apply error concealment techniques to conceal the loss of video information. If the client has some prefetched frames in its buffer ($p_l > 0$), the server retransmits the lost frames. Whenever loss occurs, the server resets its send window to $w = 1$. The loss of frames is indicative of acute link overload and by reducing the send window we can throttle the server and thus alleviate the congestion. We refer to the send window policy described in this section as the *basic window policy*. It can be summarized as follows. A connection starts with a send window of one, that is, $w_0 = 1$. The window is increased by a small increment Δw (we use $\Delta w = 0.1$) at the beginning of each frame period. The number of frames a connection is allowed to send is limited by the integral part of the send window. If loss occurs, the window is reset to one.

4 Refinements of the Decentralized Prefetching Protocol

4.1 Client Buffer Constraint

We first introduce an important modification of the decentralized prefetching protocol. This modification limits the number of b's an ongoing connection may have in its client buffer. This important refinement is useful when the client has finite buffer capacity, B . This refinement works as follows. Suppose that the server is considering transmitting frame k . It transmits this frame in the current slot only if the send window allows the transmission of the frame and the *client buffer constraint*

$$b_l + x_k \leq B \quad (3)$$

is satisfied. Condition (3) ensures that the server does not overflow the client buffer.

4.2 Dynamic Send Window

We now introduce a refinement of the send window policy. The idea behind this refinement is to increase the send window by a large increment when the client buffer holds only a small reserve of prefetched frames and throttle the server when the client buffer contains a large reserve of prefetched frames. To this end, we compute the window increment as a function of the amount of prefetched data in the client buffer:

$$\Delta w_l = \Delta w_{\max} \left(1 - \frac{b_l}{B}\right)^e, \quad \Delta w_{\max} > 0, e > 0. \quad (4)$$

When the client buffer is empty at the beginning of slot l , that is, when $b_l = 0$, the send window is incremented by Δw_{\max} . When the client buffer is full, that is, when $b_l = B$, the send window is not increased at all. We refer to this send window policy as the *dynamic window policy*. The dynamic window policy can be summarized as follows. At the beginning of slot l , the server computes Δw_l according to (4), calculates the new send window, $w_l = w_{l-1} + \Delta w_l$, and sends $\lfloor w_l \rfloor$ frames. As with the basic window policy, a new connection starts with a send window of $w_0 = 1$ and resets the window to $w_l = 1$ if the acknowledgments do not arrive by the end of slot l .

The parameters Δw_{\max} and e are used to tune the policy. We provide a detailed numerical study of the impact of these parameters on the performance of our decentralized prefetching protocol in the accompanying technical report [19]. Because of page limitations we give here only a brief discussion of these parameters. A large Δw_{\max} gives large increments Δw and thus allows the server to send more frames. The parameter Δw has to be large enough to allow for prefetching of future frames. If Δw_{\max} is too large, however, a few connections can "swamp" the multiplexer and degrade the protocols' performance.

The parameter e can be set to give a connection with a nearly empty client buffer an increased chance of filling its client buffer. To see this, note that for

$e = 1$, the window increment decreases linearly as the client buffer contents increase. For $e > 1$, connections with fairly large buffer contents are allowed substantially smaller increments (compared to when $e = 1$), while a connection with small client buffer contents has still a large window increment. This gives a connection with a small reserve of prefetched frames a better chance of filling its client buffer.

We found that the decentralized prefetching protocol works well for a wide range of parameters. In particular, Δw_{\max} values between 2 and 8 and e values between 4 and 10 give good performance [19]. We choose $\Delta w_{\max} = 5$ and $e = 6$ for the numerical work in this paper.

4.3 Randomized Transmission

In this subsection we introduce a refinement that helps to ensure fair bandwidth distribution among the ongoing connections. In the protocol described so far, the server transmits the first $\lfloor w_1 \rfloor$ frames of the video immediately after the request of the client has been processed. Subsequent transmissions are scheduled l/F seconds, $l = 1, \dots, N - 1$, after the initial transmission. The relative slot phases remain fixed for the entire duration of a video. To see how this can lead to unfair bandwidth distribution consider the phase alignment with $t_j \gg t_k$ depicted in Figure 4. Suppose connections j and k are the only connections in progress. Now

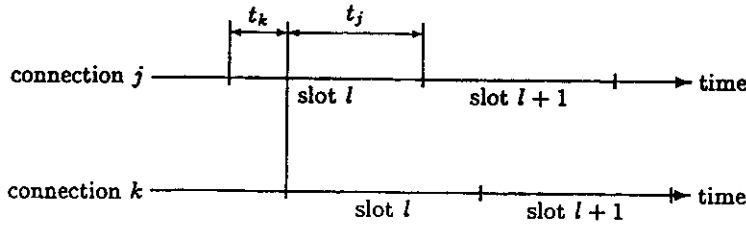


Fig. 4. Phase alignment favoring connection j . If both connections fill the multiplexer buffer to capacity whenever they transmit, connection j can transmit Rt_j bits in a frame period, while connection k can transmit only Rt_k bits

consider a scenario where connection j fills the multiplexer buffer completely at the beginning of its slot l . Connection k is then able to fit Rt_k bits into the multiplexer buffer at the beginning of its slot l . When connection j is up for transmission again, at the beginning of its slot $l+1$, it can fit Rt_j bits into the multiplexer buffer. With the depicted phase alignment ($t_j \gg t_k$), connection k has clearly a disadvantage since it can transmit only Rt_k bits in a frame period as long as connection j keeps on filling the multiplexer buffer to capacity.

To avoid this unfair bandwidth distribution we introduce *randomized transmission*: The server transmits the first $\lfloor w_1 \rfloor$ frames of the video immediately after the request of the client has been processed. The server draws a random phase δ_l , $l = 1, \dots, N - 1$ from a uniform distribution over $[-1/2F, 1/2F]$ in each frame period. The subsequent transmissions are scheduled $l/F + \delta_l$ seconds, $l = 1, \dots, N - 1$ after the initial transmission. With this transmission rule, the slot phases are constantly reshuffled. Unfair phase alignments can therefore not persist for extended periods of time.

Note that with randomized transmission, two consecutive transmissions can be spaced less than $1/F$ seconds apart. (In fact, two transmissions can be scheduled for the same time. This happens when the server draws the random phases $\delta_l = 1/2F$ and $\delta_{l+1} = -1/2F$. Note, however, that we are ignoring processing delays.) Thus, even with a maximal delay in the multiplexer of $1/F$ seconds and ignoring propagation and processing delays, the acknowledgements may not arrive before the next transmission.

We propose two solutions for this problem. The first solution relies on the multiplexer sending back an error message to the server when a frame does not fit into the multiplexer buffer. We note that the Source Quench Error Message defined in the Internet Control Message Protocol (ICMP) [20, p.160] may be used for this purpose. The server assumes that a frame is successfully received by the client if the multiplexer does not send an error message. The client is not required to send acknowledgments when this approach is used. We refer to this approach as *multiplexer feedback*.

An alternative solution is to randomly spread the transmissions not over the entire frame period but instead over half the frame period by drawing the random phases δ_l from a uniform distribution over $[-1/2F, 0]$. Setting the multiplexer buffer to $R/2F$ ensures that the acknowledgements from the client are received before the next transmission is scheduled. We refer to this approach as the *client feedback* approach. We note that by spreading out the transmissions over a smaller interval and reducing the multiplexer buffer client feedback does slightly degrade the performance of the decentralized prefetching protocol. We provide a detailed numerical study of the impact of client feedback on the protocols performance in the technical report [19].

5 Experimental Results

In this section we present the results of a simulation study of the decentralized prefetching protocol. The study is based on MPEG 1 encodings of the four movies in Table 1. The frame size traces, which give the number of bits in each video frame, were obtained from the public domain [16]. (We are aware that these are low resolution traces and some critical frames are dropped; however, the traces are extremely bursty.) The movies were compressed with the Group of Pictures (GOP) pattern IBBPBBPBBPBB at a frame rate of $F = 24$ frames/sec. Each of the traces has 40,000 frames, corresponding to about 28 minutes. The mean number of bits per frame and the peak-to-mean ratio are given in Table 1.

Trace	Mean (bit)	Peak/Mean
lambs	7,312	18.4
bond	24,308	10.1
terminator	10,904	7.3
mr.bean	17,647	13.0

Table 1. Statistics of MPEG-1 traces

We assume in our numerical work that the video frames are transported in packets consisting of 512 bytes of payload and 40 bytes of overhead. We fix the link rate at $R = 45$ Mbps; the corresponding multiplexer buffer holds 234,375 bytes ($= R/F$). We define the link utilization as the sum of the mean bit rates of all ongoing connections divided by R . In our experiments we use a mix of the the four movies that achieves 95% link utilization. Specifically, we use 55 lambs connections, 17 bond connections, 37 terminator connections, and 23 mr.bean connections. With these numbers, each of the four movies accounts for roughly one fourth of the link load.

In each realization of our simulation, we generate a random starting frame $\theta(j)$ for each of the J ongoing connections. The value $\theta(j)$ is the frame that is removed from the j th client buffer at the end of slot 1. The $\theta(j)$'s are independent and uniformly distributed over $[1, N]$. All connections start with empty client buffers at the beginning of slot 1. When the N th frame of a video is removed from a client buffer, we assume that the corresponding user immediately requests to see the entire movie again. Thus, there are always J connections in progress. For each replication of the simulation we also draw random (non-synchronized) slot phases $t(j)$ for each of the J connections. The $t(j)$'s are independent and are drawn from a uniform distribution over $[0, 1/F]$. The $t(j)$'s determine the relative starting times of the slots for the J connections. Note that the frames of connection j scheduled for transmission in slot l are placed in the multiplexer buffer at the beginning of the slot (see Figure 3), that is, server j puts its traffic into the queue at instants $t(j) + (l-1)/F$, $l = 1, \dots, N$ ($t(j) + (l-1)/F + \delta_{l-1}$, $l = 1, \dots, N$ with randomized transmission). In all our simulations we assume that all clients have the same buffering capacity, B . We allow a warm-up time of 40,000 frame periods for each replication before counting frame periods with starvation. We run each simulation until the 90% confidence interval is less than 10% of the estimated loss probability. We define the loss probability as the long run fraction of frame periods for which at least one client experiences starvation.

Figure 5 shows the performance of our basic decentralized prefetching protocol, and its various refinements. We plot the loss probability as a function of the client buffer size for 95% link utilization. For the basic window policy we use a fixed window increment of $\Delta w = 0.1$. The parameters of the dynamic window policy are set to $\Delta w_{\max} = 5$ and $e = 6$. The figure shows that the basic window policy has unacceptably high losses. The loss probability is about 8×10^{-3} for 1 MByte of client buffer. We also see that the dynamic window policy brings significant improvement over the basic window policy. The loss probability for

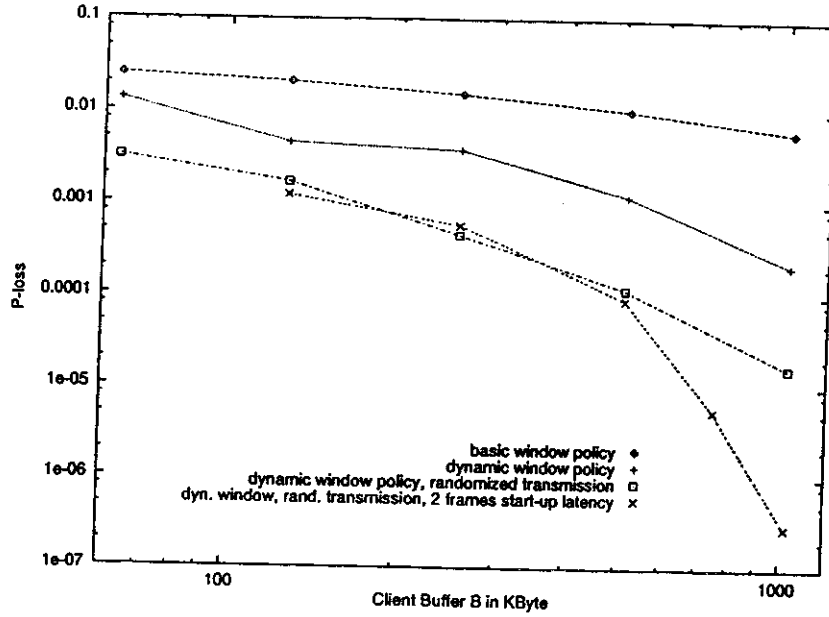


Fig. 5. Loss probability as a function of client buffer size for the basic decentralized prefetching protocol and its refinements

the dynamic window policy is almost one order of magnitude smaller. Adding randomized transmission further reduces the loss probability significantly. The loss probability for the dynamic window policy with randomized transmission for 1 MByte of client buffer is about 1.5×10^{-5} . We employ multiplexer feedback here. By allowing a short start-up latency of 2 frame periods we can further reduce the loss probability significantly.

In Figure 6 we compare our decentralized prefetching protocol with Join-the-Shortest-Queue (JSQ) Prefetching [12] and Optimal Smoothing [17, 11, 18]. The plot gives the loss probability as a function of the client buffer size for 95% link utilization. The optimal smoothing curves are obtained by applying the optimal smoothing algorithm [17, 11, 18] to the traces used for the simulation of the prefetch policy. We then compute the loss probability for statistically multiplexing the smoothed traces on a bufferless 45 Mbps link with the Large Deviation approximation [9, 11]. The large deviation approximation is known to be highly accurate [9, 21]. We do this for two versions of optimal smoothing: no initiation delay and a 10 frame initiation delay [8, 11, 22]. The decentralized prefetching results are for the dynamic window policy with randomized transmission, multiplexer feedback and 2 frames start-up latency. The JSQ prefetching results are from [12]. Decentralized prefetching clearly outperforms optimal smoothing, both without and with start-up latency. The loss probability for decentralized prefetching is over one order of magnitude smaller than the loss probability for

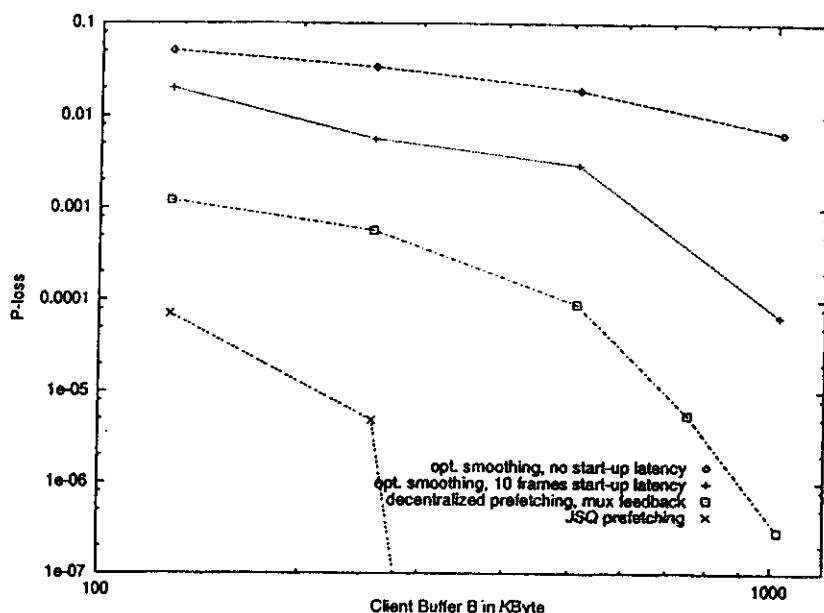


Fig. 6. Loss probability as a function of client buffer size for optimal smoothing, decentralized prefetching and JSQ prefetching

optimal smoothing with start-up latency. The gap widens to over two orders of magnitude for 1 MByte of client buffer.

References

1. I. Dalgic and F. A. Tobagi. Characterization of quality and traffic for various video encoding schemes and various encoder control schemes. Technical Report CSL-TR-96-701, Stanford University, Departments of Electrical Engineering and Computer Science, August 1996.
2. D. Wrege, E. Knightly, H. Zhang, and J. Liebeherr. Deterministic delay bounds for VBR video in packet-switching networks: Fundamental limits and tradeoffs. *IEEE/ACM Transactions on Networking*, 4(3):352-362, June 1996.
3. E. W. Knightly and H. Zhang. Traffic characterization and switch utilization using a deterministic bounding interval dependent traffic model. In *Proceedings of IEEE Infocom '95*, Boston, MA, April 1995.
4. J. Liebeherr and D. Wrege. Video characterization for multimedia networks with a deterministic service. In *Proceedings of IEEE Infocom '96*, San Francisco, CA, March 1996.
5. W. Feng, F. Jahanian, and S. Sechrest. Providing VCR functionality in a constant quality video-on-demand transportation service. In *IEEE Multimedia*, Hiroshima, Japan, June 1996.
6. W. Feng and J. Rexford. A comparison of bandwidth smoothing techniques for the transmission of prerecorded compressed video. In *Proceedings of IEEE Infocom*, Kobe, Japan, April 1997.

7. J.M. McManus and K.W. Ross. Prerecorded VBR sources in ATM networks: Piecewise-constant rate transmission and transport. In *Proceedings of SPIE*, Dallas, TX, October 1997. Available at <http://www.seas.upenn.edu/~ross/>.
8. J. Salehi, Z.-L. Zhang, Kurose J, and D. Towsley. Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing. Technical report, University of Massachusetts, 1995.
9. M. Reisslein and K. W. Ross. Call admission for prerecorded sources with packet loss. *IEEE Journal on Selected Areas in Communications*, 15(6):1167–1180, August 1997.
10. M. Grossglauser, S. Keshav, and D. Tse. RCBR: A simple and efficient service for multiple time-scale traffic. In *ACM SIGCOMM*, 1995.
11. Z. Zhang, J. Kurose, J. Salehi, and D. Towsley. Smoothing, statistical multiplexing and call admission control for stored video. *IEEE Journal on Selected Areas in Communications*, 13(6):1148–1166, August 1997.
12. M. Reisslein and K. W. Ross. A join-the-shortest-queue prefetching protocol for VBR video on demand. In *IEEE International Conference on Network Protocols*, Atlanta, GA, October 1997. Available at <http://www.seas.upenn.edu/~reisslei/>.
13. J.M. McManus and K.W. Ross. A comparison of traffic management schemes for prerecorded video with constant quality service. Available at <http://www.seas.upenn.edu/~ross>.
14. V. Jacobson. Congestion control and avoidance. In *Proceedings of SIGCOMM '88 Symposium*, August 1988.
15. L. Brakmo and L. Peterson. TCP Vegas: end to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, October 1995.
16. O. Rose. Statistical properties of MPEG video traffic and their impact on traffic modelling in ATM systems. Technical Report 101, University of Wuerzburg, Institute of Computer Science, Am Hubland, 97074 Wuerzburg, Germany, February 1995.
ftp address and directory of the used video traces:
<ftp://ftp-info3.informatik.uni-wuerzburg.de/pub/MPEG/>.
17. J. Salehi, Z. Zhang, J Kurose, and D. Towsley. Optimal smoothing of stored video and the impact on network resource requirements. *submitted to IEEE/ACM Transactions on Networking*, 1996.
18. J. Rexford and D. Towsley. Smoothing variable-bit-rate video in an internetwork. Technical Report CMPSCI-97-33, University of Massachusetts at Amherst, Department of Computer Science, May 1997. Available via <ftp://gaia.cs.umass.edu/pub/Rex97:Tandem.ps.Z>.
19. Martin Reisslein, Keith W. Ross, and Vincent Verillotte. A decentralized prefetching protocol for VBR video on demand (extended version). Technical report, University of Pennsylvania, Department of Systems Engineering, November 1997. Available at <http://www.seas.upenn.edu/~reisslei/>.
20. R. Stevens. *TCP/IP Illustrated, Volume 1, The Protocols*. Addison-Wesley, 1994.
21. A. Elwalid, D. Mitra, and R. H. Wentworth. A new approach for allocating buffers and bandwidth to heterogeneous regulated traffic in an ATM node. *IEEE Journal on Selected Areas in Communications*, 13(6):1115–1127, August 1995.
22. J. Dey, S. Sen, J. Kurose, D. Towsley, and J. Salehi. Playback restart in interactive streaming video applications. In *To appear in Proceedings of IEEE Multimedia*, Ottawa, Canada, 1997.