A Join–the–Shortest–Queue Prefetching Protocol for VBR Video on Demand*

Martin Reisslein and Keith W. Ross†
Department of Systems Engineering
University of Pennsylvania
Philadelphia, PA 19104
{reisslei, ross}@seas.upenn.edu
http://www.seas.upenn.edu/~{reisslei, ross}

Abstract

We present a high-performance prefetching protocol for the delivery of prerecorded VBR video from a server across a packet–switched network to a large number of clients. Not only does the protocol give constant perceptual quality and almost 100% link utilization, but it also allows for immediate commencement of the video upon user request and near instantaneous response to pause/resume and temporal jumps. The protocol requires (1) that each client have a small amount of memory dedicated to the application (2) that there is one bottleneck shared link between the server and the clients. Our protocol is based on the observation that there are frequent periods of time during which the shared link's bandwidth is under utilized. During these periods the server can prefetch frames from any of the ongoing videos and send the frames to the buffers in the appropriate clients. The server chooses prefetched frames according to a join–the–shortest–queue policy. We present simulation results of our prefetch policy that are based on MPEG encoded traces.

1 Introduction

We present a high-performance prefetching protocol for the delivery of video on demand (VoD) from a server across a packet-switched network to a large number of clients. The protocol assumes that the videos resident on the video server are variable-bit-rate (VBR) encoded. Not only does this protocol give constant perceptual quality and almost 100% link utilization, but it also allows for immediate commencement of the video upon user request and near instantaneous response to interactive actions (pause/resume and temporal jumps).

*Supported partially by NSF grant NCR96–12781

†Corresponding Author: Keith W. Ross, phone: (215) 898-6069, fax: (215) 573-2065

To achieve this high performance our protocol has two requirements. First we require all of the clients to have a small amount of memory dedicated to the VoD application. Second, as shown in Figure 1, we require that there be at most one bottleneck shared link between the video server and the clients. If the clients are connected to an ADSL residential access network, then this second requirement can be achieved by attaching the video server directly to the ADSL central office; in this case the shared link is the link from the server to the ADSL central office. If the clients are connected to cable, then the second requirement can be achieved by attaching the video server directly to the cable headend.

The user's client could be a television with a set-top box capable of performing buffering and decoding, or it could be a household PC. The video server could be a cache containing the week's most popular videos. A central repository could multicast the videos to this and other caches over the Internet using the traditional best–effort service.

Our protocol explicitly assumes that the videos are VBR encoded with high peak-to-mean ratios. If the videos were instead CBR encoded, it would be relatively straightforward to design a network transport scheme that gives immediate playback, near instantaneous response to interactive actions, and high link utilization. However, the CBR encoding causes the perceived image quality to vary, which is unacceptable for high-action content such as action movies, airline pilot training, and sporting events. For these latter services, it is preferable to perform open-loop VBR encoding or constant-quality VBR encoding.

Our protocol achieves the constant perceptual quality, responsiveness to user interactivity, and high link utilizations by exploiting two special properties of the prerecorded video: (1) for each video, the traffic in

0-8186-8061-X/97 $10.00

each video frame is known before the video session begins; (2) while the video is being played, some of the video can be prefetched into the client memory. It is this second property – the ability to prefetch a portion of any video – that is particularly central to our high-performance protocol.

Our protocol is based on the observation that, due to the VBR nature of the multiplexed traffic, there will be frequent periods of time during which the shared link's bandwidth is under utilized. During these periods the server can prefetch video frames from any of the ongoing videos and send the prefetched frames to the buffers in the appropriate clients. With this prefetching, many of the clients will typically have some prefetched reserve in their buffers. Our protocol also specifies the policy for how the server selects the prefetched frames. This policy is the join-the-shortest-queue (JSQ) policy, which can be roughly described as follows: within each frame time the server repeatedly selects frames from the connections that have the smallest number of prefetched frames in their client buffers. The JSQ policy creates a *buffer pooling effect* so that the system behaves as if the individual client buffers are aggregated into one large buffer which is shared by all the clients. Our empirical work with public-domain traces indicates that prefetching combined with the JSQ policy gives dramatic reductions in packet loss. In particular, if each client dedicates a small amount of buffer capacity to the VoD application, this scheme can multiplex a large number of connections over the shared link and have negligible playback starvation.

With JSQ prefetching, through buffer pooling, the connections collaborate in order to minimize the overall packet loss. This collaboration among the connections contributes significantly to the protocol's outstanding performance. We also present numerical results which show that *Optimal Smoothing*, a non-collaborative prefetching policy, can have packet loss that is several orders of magnitude higher than that of JSQ prefetching for a wide range of buffer sizes.

1.1 Literature Review

The papers on the management of prerecorded VBR traffic in the network fall into three categories: deterministic; deterministic with smoothing and/or prefetching; and probabilistic. The principal performance metrics for all of these schemes are average link utilization, initial delay, delays after interactive actions, and client buffer size.

The deterministic schemes send into the network the original VBR traffic, and admission control ensures that the delays never exceed a prespecified limit [1] [2] [3] [4] [5]. For highly variable VBR traffic, these deterministic schemes typically require large initial delays to achieve moderate link utilizations [5].

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. Admission control can then be based on the peak-rate of the smoothed trace. CRTT [6] is the extreme case of such a scheme, whereby the server transmits packets into a reserved CBR connection at the average rate of the video; such CBR connections are available from ATM and are being proposed for the Internet. Although CRTT produces close to 100% link utilizations, it has potentially a long initial delay and potentially long delays after temporal jumps. 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 [8] [4] [9] [10] [11]. As compared with CRTT, these schemes trade off link utilizations and simplicity for lower initial delay and lower interactivity delays. In summary, none of the deterministic schemes (with or without prefetching and smoothing) allows for both high link utilizations and consistently high responsiveness (less than a few seconds) to interactivity.

For the probabilistic approaches, [12] 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 [12] we developed an admission control procedure based on a large-deviations approximation. In [8] and [13] related ideas are explored whereby the original traffic is first smoothed before it is statistically multiplexed at an unbuffered link; the statistically multiplexing of the smoothed traffic can substantially increase link utilization at the expense of small packet loss probabilities. All of these hybrid schemes perform non-collaborative prefetching; because JSQ is collaborative, it can have significantly better performance than these hybrid schemes.

2 Architecture Description

Figure 1 illustrates our basic model for VoD. The video server contains a large number of videos 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. Although some of the connections

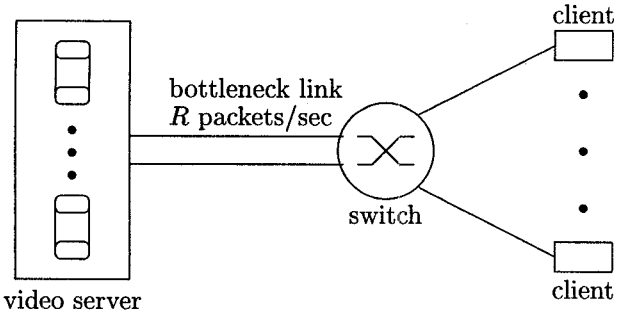

Figure 1: Prerecorded videos multiplexed over a link of capacity R packets/sec.

might be transmitting the same video, the phases (i.e., the start times) are typically different. The server packetizes the frames of the ongoing connections and then statistically multiplexes and transmits the packets into its link; for simplicity, we assume throughout that the packets are of fixed length. Let $x_n(j)$ denote the number of packets in the nth frame of the jth connection. Because the videos are prerecorded, $(x_1(j), x_2(j), \ldots, x_N(j))$ is fully known at connection establishment.

When a client requests a specific video, the server makes an admission control decision by deciding whether or not to grant the request. If it grants the request, a connection is established and the server immediately begins to transmit the connection's packets into the network. The connection's packets are transmitted in a fixed, predetermined order. When packets arrive at the client, they 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 the prefetch 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.

We denote R (in packets/sec) for the maximum transmission rate of the server. Although our protocol allows for pause and temporal jumps, we will initially exclude these interactive features in order to simplify the discussion; thus N/F seconds elapse from when the user begins to watch the video until when the video ends.

To make these ideas a little more precise, we divide time into slots of length $1/F$. Let $p_l(j)$ be the number of frames in the prefetch buffer for connection j at the beginning of slot l. Let $\Delta_l(j)$ be the number of frames of connection j that arrive to the prefetch buffer during the the lth slot. At the end of each slot, one frame is removed from each prefetch buffer that has one or more frames. Thus

$$p_{l+1}(j) = [p_l(j) + \Delta_l(j) - 1]^+, \qquad (1)$$

where $[x]^+ := \max(x, 0)$. Denote $\theta_l(j)$ for the frame number of connection j that is supposed to be removed at the end of slot l; thus $\theta_{l+1}(j) = \theta_l(j) + 1$.

During each slot of length $1/F$ seconds the server must decide which frames to transmit from the J ongoing videos. The *prefetch policy* is the rule that determines which frames are transmitted in each slot. The maximum number of packets that can be transmitted in a slot is R/F (which for simplicity we assume to be an integer).

For each ongoing connection j the server keeps track of the the prefetch buffer contents $p_l(j)$; this can be done through the recursion (1) without communicating with the client. Initially, we require the server to skip the transmission of the entire frame $\theta_l(j)$ whenever $p_l(j) + \Delta_l(j) = 0$. Thus, the server does not transmit a frame that will not meet its deadline at the client.

Before defining our prefetch policy, it is useful to introduce some more notation. Let $b_l(j)$ be the number of packets in the prefetch buffer for connection j at the beginning of slot l. Let $\Delta'_l(j)$ denote the number of packets of connection j that arrive to the prefetch buffer during the the lth slot. These definitions imply

$$b_{l+1}(j) = [b_l(j) + \Delta'_l(j) - x_{\theta_l(j)}(j)]^+. \qquad (2)$$

3 The JSQ Prefetch Policy

Our Join-the-Shortest-Queue (JSQ) prefetch policy attempts to balance the number of frames across all of the prefetch buffers. In describing this policy, we drop the subscript l from all notations. At the beginning of each slot the server determines the j^* with the smallest $p(j)$, transmits one frame from connection j^* and increments $p(j^*)$. Within this same slot the server repeats this procedure over and over again, at each iteration finding a new j^* that minimizes $p(j)$, transmitting a frame from connection j^* and incrementing $p(j^*)$.

Due to the finite transmission rate of the server, at some point the server must stop transmitting frames within the slot. To this end, let z be a variable that keeps track of the total number of packets sent within

the slot; z is reinitialized to zero at the beginning of every slot. The stopping rule works as follows. Before transmitting a frame from connection j^* we check to see if

$$z + x_{\sigma(j^*)}(j^*) \leq R/F, \tag{3}$$

where $\sigma(j^*)$ is the frame of connection j^* that is being considered for transmission. If this condition holds, then we transmit the frame and update z; otherwise, we do not transmit the frame, set $p(j) = [p(j) - 1]^+$ for $j = 1, \ldots, J$ and recommence the procedure for the subsequent slot. This is our *basic stopping rule*; later we shall discuss a slightly more complicated stopping rule.

With prefetching, it is possible that all of a connection's frames have been transmitted but not all of its frames have been displayed. When a connection reaches this state, we no longer consider it in the above JSQ prefetching policy. From the server's perspective, it is as if the connection has been terminated.

3.1 Refinements of the JSQ policy

We now discuss a few important refinements of the JSQ policy. First, we introduce a refined stopping rule. Recall that during each slot the server transmits a sequence of frames until condition (3) is violated; once (3) is violated, the server does not transmit any more frames in the slot. An alternative stopping rule is to try to transmit more frames in the slot by removing from consideration the connection that violates (3) and finding a new j^* that minimizes $p(j)$. If the condition (3) holds with the frame from the new connection j^*, we transmit the frame, update $p(j^*)$, and continue the procedure of transmitting frames from the connections that minimize the $p(j)$'s. Whenever a frame violates condition (3), we skip the corresponding connection and find a new j^*. When we have skipped over all of the connections, we set $p(j) = [p(j) - 1]^+$ for $j = 1, \ldots, J$ and move on to the next slot. This is our *refined stopping rule*. To reduce the online computational effort we can also, of course, consider rules which fall between the basic and refined stopping rules. For example we could use a rule which stops when condition (3) has been violated K times where $1 < K < J$.

The next refinement of the JSQ policy limits the number of packets an ongoing connection may have in its client's prefetch buffer. This important refinement is useful when the client for connection j, $j = 1, \ldots, J$, has finite buffer capacity $B(j)$. This refinement works as follows. Suppose that the server is considering transmitting frame $\sigma(j^*)$ from connection j^*. Let $b(j^*)$ be the current number of packets in the prefetch buffer for connection j^*. It transmits this frame in the current slot only if condition (3) and the condition

$$b(j^*) + x_{\sigma(j^*)}(j^*) \leq B(j) \tag{4}$$

are satisfied. Condition (4) ensures that the server does not overflow the prefetch buffer for connection j^*. With this additional condition, we extend the definitions of the stopping rules in the obvious way.

3.2 System Dynamics and Pooling

We now crudely describe the dynamics of the prefetch buffer contents. Let us make the assumption that whenever all N frames of a connection are displayed, the same user immediately requests a new connection; thus, with this assumption there are always J videos in progress. Let us make the realistic assumption that

$$\sum_{j=1}^{J} x_{avg}(j) < R/F, \tag{5}$$

where $x_{avg}(j)$ is the average number of packets in a frame in the jth connection. The condition (5) says that the long-run average aggregate consumption rate of the ongoing videos is less than the maximum server supply rate.

The JSQ prefetch policy will make most of the $p(j)$'s nearly equal to each other. Moreover, because of (5) there will be a general tendency for each of the $p(j)$'s to increase. In other words, there is typically a "pack" of $p(j)$'s with near equal values, with each $p(j)$ in the pack drifting towards $B(j)$. A $p(j)$ can separate itself from the pack if it hits $B(j)$ while the pack continues to gain higher occupancies. It will also break from the pack if the corresponding connection reaches the state of having all its frames transmitted. In this case, the $p(j)$ will descend by one in each slot until it hits zero; when it hits zero, $p(j)$ will quickly return to the pack.

One important feature of the JSQ prefetching policy is that it creates a *pooling effect*, namely, the individual buffers act as a large collected buffer of capacity $B(1) + \cdots + B(J)$. To see this, suppose that a large number of connections start simultaneously (therefore having few frames in their prefetch buffers) and the remaining connections have a large number of frames in their prefetch buffers. Also assume that the aggregate consumption rate temporarily exceeds R packets/sec. Then if the consumption rate across the connections that have just started is less than R, the JSQ policy will prevent frame loss as long as the high aggregate consumption rate does not persist for too long. This is because the prefetch policy will feed the connections that have just started until their prefetch

buffers catch up with rest of the pack. Thus, the likelihood of cell loss in the near future typically depends on the $p(j)$'s only through the pooled buffer content $p(1) + p(2) + \cdots + p(J)$.

3.3 Experimental Results

In this subsection we present the results of a simulation study for the JSQ prefetch policy described in the previous section. Throughout we use the refined stopping rule discussed in Section 3.1. Our simulation study makes use of MPEG 1 encodings of the four movies in Table 1. The associated traces, obtained from the public domain [14], give the number of bits in each frame. (We are aware that these are low resolution traces and some critical frames are dropped; however, the traces are extremely bursty. We have obtained similar results, not reported here, for *Star Wars* and *Oz*.) Each of the movies was compressed with the 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. Table 1 also gives the average size of I, P and B frames and the percentage of encoding information carried in I, P and B frames. It can be argued that the average rate in bits/sec is lower than what we would expect for digital compressed video (e.g., MPEG-2 video); for this reason, we have chosen a relatively small server transmission rate of 45 Mbps. We expect VoD systems in the future to have videos with larger average rates and a proportionally larger server transmission rate. In this scaling, the number of videos that can be multiplexed will be approximately constant. We furthermore assume that each packet consists of 512 bytes of payload and 40 bytes of overhead; therefore, $R = 81,521$ packets/sec.

We define the link utilization as the average number of packets per second, summed over all connections in progress, divided by R. In our experiments we use various mixes of the the four movies. Each of the mixes has a different link utilization. Our 90% link utilization consists of 52 lambs connections, 16 bond connections, 35 terminator connections, and 22 mr.bean connections. Our 95% link utilization consists of 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 jth prefetch buffer at the end of slot 1. The $\theta(j)$'s are independent and uniformly distributed over $[1, N]$. All connections start with empty prefetch buffers at the beginning of slot 1. When the Nth frame of a video is removed from a prefetch buffer, we assume that the corresponding user immediately requests to see the entire movie again. Thus, there are always J connections in progress.

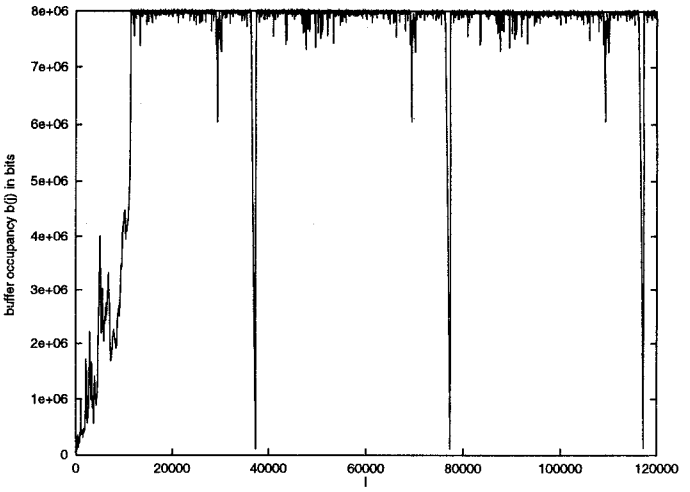

Figure 2: Prefetch buffer contents in a 1 Mbyte buffer in bits for a lamb's connection.

In Figure 2 we set the buffer capacity of each client to 1 Mbyte and the link utilization to 95%. The figure shows the prefetch buffer contents in bits for one of the lamb's connections over 120,000 simulated frame periods; the initial starting random phase for this connection is $\theta = 2,689$. At $l = 36,079$ frame periods all of the movie's frames have been transmitted to the prefetch buffer; this is why the buffer content drops to zero at $l = 37,311$, when the movie ends and starts over for the user. Note that when the movie restarts at $l = 37,311$ the JSQ prefetch policy fills the prefetch buffer for the movie in just a few frame periods. Also note that the buffer contents typically hug the 1 Mbyte buffer limit. Note that the buffer occupancy is nearly periodic, with period equal to the length of the movies (40,000 frames).

In Figure 3 we plot the 90 % confidence intervals of $P_{\text{loss}}^{\text{time}}$ for the prefetch buffer sizes ranging from 2 KBytes to 256 KBytes. $P_{\text{loss}}^{\text{time}}$ is the long–run fraction of frame periods for which loss occurs for at least one of the J ongoing videos; see [12] for more details. Each of the confidence intervals is based on 1000 replications. On this figure $P_{\text{loss}}^{\text{time}}$ is plotted for both 95% and 90% utilizations.

Figure 3 shows the dramatic improvement in performance that comes from prefetching and the JSQ policy. Without any prefetching we have $P_{\text{loss}}^{\text{time}} = 0.29$

Trace	Mean bits	Peak/ Mean	Avg. Frame Size			% of info		
			I	P	B	I	P	B
lambs	7,312	18.4	38,025	7,436	3,424	43	25	32
bond	24,308	10.1	83,293	41,427	10,513	29	42	29
terminator	10,904	7.3	37,387	14,120	6,387	29	32	39
mr.bean	17,647	13.0	75,146	18,275	10,215	35	26	39

Table 1: Statistics of MPEG-1 traces.

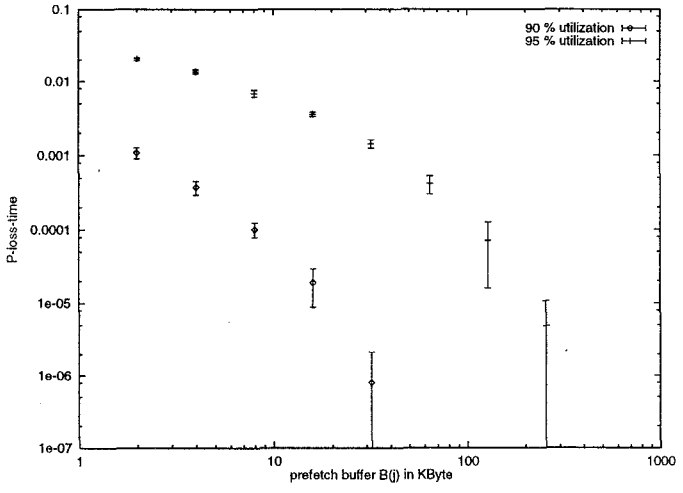

Figure 3: Confidence intervals of $P_{\text{loss}}^{\text{time}}$ for link utilizations of 90 % and 95 %. $P_{\text{loss}}^{\text{time}}$ is the fraction of frame times during which loss occurs.

for 95 % utilization. By increasing the capacity only to 256 KBytes, the loss probability is reduced to $P_{\text{loss}}^{\text{time}} \approx 4.7 \times 10^{-6}$. By further increasing the buffer to 512 KBytes, no loss was observed for any of 8000 replications, corresponding to 3.2×10^8 frame periods! Similarly, for 90 % utilization we did not observe any loss for 1000 replications, corresponding to 4×10^7 frame periods, for 64 KByte prefetch buffer. Whenever loss occurred, the buffer contents at each of the clients was nearly zero, confirming the existence of a strong pooling effect. (The existence of pooling is further justified by the analytical work of Reiman [15]).

Figure 3 demonstrates that with a small prefetch buffer at each client the JSQ prefetching policy will allow for almost 100% utilization with negligible packet loss. Also note that the scheme allows for near immediate playback of the video upon user request. It can be argued that for MPEG-2 traces with an order of magnitude larger average rate, the prefetch buffer will have to be an order of magnitude larger to achieve the same loss probabilities. But even with this order of magnitude increase, only 5 Mbytes of prefetch buffer is required to give negligible packet loss.

3.4 Efficient Implementation of the JSQ Prefetching Policy

In this subsection we describe a list based approach that allows for efficient implementation of the JSQ prefetch policy. We found in our experiments that it takes only 2.5 msec to execute the administrative steps necessary in each frame period for 132 video connections on a SPARCstation 2. We applied the refined stopping rule in these experiments and took also the buffer capacity test (4) into account.

The underlying data structure in our implementation is a singly linked list. Each list element is a record storing the data pertaining to an active connection; in particular, we keep track of $\theta(j)$, $p(j)$ and $b(j)$. We also keep track of an index indicating which video is being transmitted. Finally, each record contains a pointer to the next list element. The list is maintained such that the index $p(j)$ is ascending as one moves down the list, that is, the connection with the smallest number of prefetched frames is on the top. In each frame period we start by considering the connections at the top of the list. We first check whether conditions (3) and (4) are satisfied. If theses conditions are satisfied, we transmit the frame, increment $p(j)$ and adjust $b(j)$. Next, we check whether the successor in the list has a smaller $p(j)$. If not, we try to prefetch the next frame. This is repeated until $p(j)$ is larger than that of the next connection in the list. At that point we have to rearrange the pointer references in order to maintain the order of the list. After the order has been restored we start over, trying to prefetch for the connection on top of the list. If we find at any point that (3) or (4) is violated, we skip the connection and try to prefetch for the successor in the list, that is, we prefetch no longer for the connection on top of the list, but move down the list as we skip over connections.

4 Interactivity

In this section we adapt the JSQ prefetch policy to account for pauses as well as forward and backward temporal jumps. Our protocol allows these interactive actions to be performed with minimal delay. We assume that whenever a user invokes a interactive action, the client sends a message indicating the interactive action to the server.

Suppose that the user for connection j pauses the movie. Upon receiving notification of the action, the server can simply remove connection j from consideration until it receives a resume message from the client; while the connection is in the paused state, its prefetch buffer remains at a constant level. A slightly more complex policy would be to fill the corresponding buffer with frames once all the other prefetch buffers are full or reach a prespecified level.

Suppose that the user for connection j makes a temporal jump of δ frames into the future. If $\delta < p(j)$, we discard δ frames from the head of the prefetch buffer and set $p(j) = p(j) - \delta$. If $\delta \geq p(j)$ we set $p(j) = 0$ and discard all the frames in the prefetch buffer. Finally, suppose that the user for connection j makes a backward temporal jump. In this case we set $p(j) = 0$ and discard all frames in the prefetch buffer.

In terms of frame loss, pauses actually improve performance because the movies which remain active have more bandwidth to share. Frequent temporal jumps, however, can degrade performance since prefetch buffers would be frequently set to zero. Whenever we set a prefetch buffer to zero, the pool loses some of its total savings, thereby increasing the likelihood of loss.

We now present some numerical results for interactive actions. We consider only forward and backward temporal jumps and ignore pauses as pauses can only improve performance; we furthermore assume that $\delta > p(j)$ for all forward temporal jumps. Our results give therefore a conservative estimate of the actual performance. In our simulation, we assume that each user performs temporal jumps repeatedly, with the time between two successive jumps being exponentially distributed with constant rate. When a user performs such an action, her prefetch buffer is set to zero. Figure 4 shows the 90 % confidence interval for $P_{\text{loss}}^{\text{time}}$ for 11, 22, 32 and 43 temporal jumps per hour (on average). This experiment was conducted with $B(j) = 256$ KBytes and a 95 % link utilization. As we would expect, loss probabilities increase as the rate of temporal jumps increase; however, the increase is not significant for a sensible number of temporal jumps.

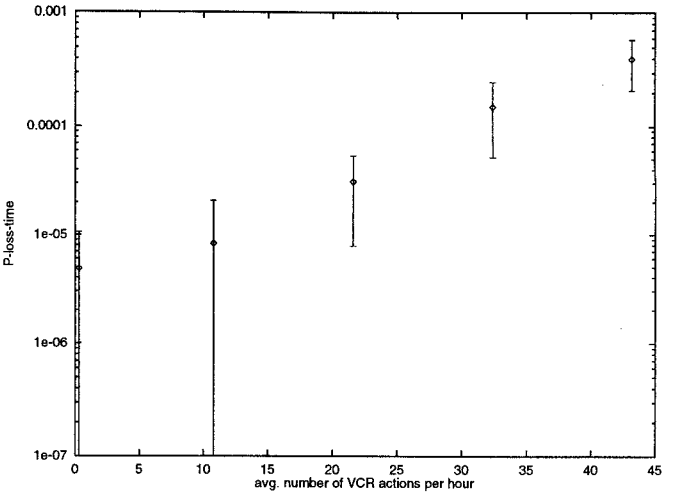

Figure 4: $P_{\text{loss}}^{\text{time}}$ as a function of the average number of temporal jumps per hour for 256 KByte buffers and 95 % link utilization.

5 Comparison between JSQ Prefetching and Optimal Smoothing

As mentioned in the Introduction, there are other protocols in the literature for the transport of VBR-encoded video which exploit prefetching and client buffers [8] [6] [9] [4] [10] [11]. In all of these other schemes, the transmission schedule for a given video is determined off-line and depends only on the traffic characteristics of that video; thus, the transmission schedule of a connection does not take into account the traffic requirements of the other connections in progress. For this reason, we refer to all the schemes cited just above as *non-collaborative* prefetching schemes.

On the other hand, the JSQ prefetching protocol determines the transmission schedule of a connection on–line, as a function of the buffer contents at *all* of the clients. For this reason, we refer to JSQ as a *collaborative prefetching scheme.* The purpose of this section is to show that the JSQ buffer fill policy is responsible for the outstanding performance reported in Section 3.3. To this end we shall compare JSQ prefetching with one of the non-collaborative schemes in the existing literature, namely, Optimal Smoothing [4, 13]. We apply the Optimal Smoothing algorithm to the traces used for the JSQ experiments (see Section 3.3). We then statistically multiplex the smoothed traces on a bufferless 45 Mbps link and calculate $P_{\text{loss}}^{\text{time}}$ using the Large Deviation approximation described in [12] and [13]. We do this for two versions of optimal smoothing: no initiation delay and a 10 frame initiation delay [4] [13] [16]. We chose Optimal Smoothing for our com-

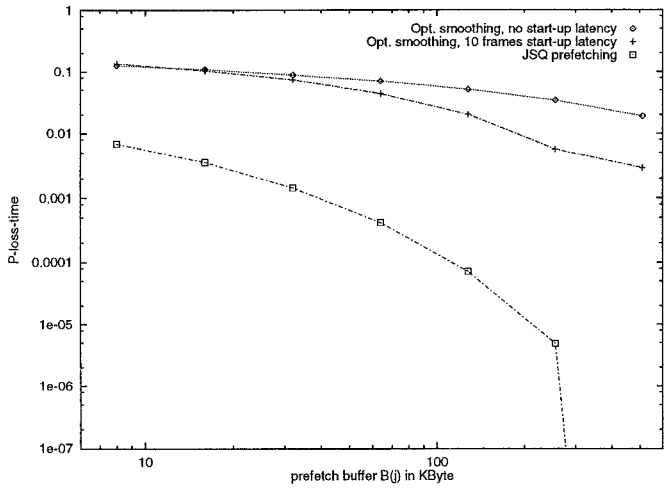

Figure 5: $P_{\text{loss}}^{\text{time}}$ as a function of buffer size for JSQ prefetching and statistical multiplexing of optimally smoothed traces for 95% average link utilization.

parison, as optimal smoothing minimizes the peak rate and variability of the smoothed traffic for a given client buffer.

The results are given in Figure 5; only the point estimates of the simulation results are plotted here. The figure shows $P_{\text{loss}}^{\text{time}}$ as a function of buffer size at the client for 95% average link utilization. The figure shows that JSQ prefetching gives a $P_{\text{loss}}^{\text{time}}$ that is more than two orders of magnitude smaller than Optimal Smoothing for a 128 Kbyte buffer, and more than three orders of magnitude smaller for a 256 Kbyte buffer. Furthermore, for this case of 95% utilization, Optimal Smoothing has unacceptably high loss probabilities for all buffer sizes shown, whereas JSQ prefetching gave no loss in 4×10^8 simulated frame periods for a buffer size of 300 Kbytes. (To account for this point on the graph, we suppose that loss occurs for one of the 4×10^8 frame periods at 300 Kbytes.) We are thus led to the conclusion that the collaborative nature of JSQ prefetching contributes significantly to its high performance. JSQ prefetching is inspired by the least-loaded routing algorithm for circuit-switched loss networks, which is known to give excellent performance and to be extremely robust over a wide range of traffic conditions [17].

We furthermore note that the JSQ policy allows for instantaneous interactive actions, making the scheme amenable to highly interactive multimedia applications. These interactions cause only a modest drop in performance (see Section 4). An indepth study of interactivity for Optimal Smoothing is given in [16]. In general, no smoothing scheme can always give instantaneous playback after all temporal jumps, because for some playback points a build-up delay is required to ensure no starvation in the future. In [16] an algorithm is given for which the build-up delay is typically very small for the traces considered. However, this build-up delay typically increases with the client buffer size, and at playback points before long high-bandwidth scenes the delay can be large (an example is given for which the delay after a temporal jump can be as much as 63 seconds for a 4Mbyte buffer). Thus, with Optimal Smoothing, as the client buffers become larger, the link utilizations increase but interactivity performance decreases. In contrast, with JSQ prefetching, both link utilization and interactivity performance improve with increasing client buffer.

We now provide an intuitive explanation of why packet loss is less of JSQ prefetching. The JSQ policy strives to (1) equalize the number of prefetched frames over all ongoing connections and (2) keep the buffers always filled. Optimal Smoothing, on the other hand, first fills the buffer when a connection starts up. The buffer is then emptied, then filled again and so on; the buffer content thus oscillates between empty and full.

Now consider a situation where a number of connections have been in progress for a while and a new connection becomes active. The JSQ policy dedicates the entire link capacity temporarily to the new connection so as to bring the number of prefetched frames up to the level of the other connections. These other connections are fed from their prefetch buffers during this period. For loss to occur, the prefetch buffers of the older connections need to be drained completely and the aggregate rate of the ongoing connections has then to exceed the link capacity. This is a highly unlikely scenario given that the prefetch buffers typically hold a large number of prefetched frames. With optimal smoothing, however, the new connection gets only a small fraction of the link capacity since the other connections have to transmit according to their fixed schedule. Losses are therefore more likely, particularly for movies requiring a large amount of bandwidth in the beginning.

Next, consider a situation where a number of connections have been active for a while and high action scenes in the movies collude and thus cause temporarily an excessive demand for bandwidth. Since JSQ prefetching keeps the buffers always full while the optimal smoothing policy drives the buffer contents periodically up and down, the aggregate buffer contents of all the ongoing connections is larger with JSQ prefetching. The JSQ policy can therefore support longer periods of excessive demand for bandwidth.

We conclude this section by mentioning that the

non-collaborative prefetching schemes do have an important advantage over our JSQ prefetching protocol – namely, they can be implemented over multiple, decentralized servers, and they can be easily adapted to run over a network with multiple bottleneck links. In particular, the server does not need to be attached to a cable headend or an ADSL switch, but can instead be distributed and placed deeper into the network. In our current research, we are adapting the JSQ prefetching protocol so that the server can be distributed and placed deeper into the network.

6 An Admission Policy for MPEG

We now present a connection admission policy that combines JSQ prefetching with Selective Discard. Because of page limitations we provide here only a very brief description of selective discard; see [18] for a detailed definition. Selective Discard works roughly as follows. When loss in a frame period is unavoidable, that is, when the sizes of the current frames of all connections with empty buffers add up to more than R/F, we discard B and P frames and transmit as many I frames as possible. Observe that with this selective discard policy the loss probability for I frames will be less than that when the clients have no buffer and the traffic in the P and B frames is equal to zero. For this conservative model, we can adapt the theory in [12] to construct a conservative large-deviation approximation for the loss of I frames. Using this approximation, we only admit a new connection if (1) the fraction of frame periods which have I-frame loss remains less than some minute number, say, 10^{-9}, and (2) the link utilization remains below, say, 95 %. This admission policy essentially guarantees no loss of I frames while simultaneously ensuring a low probability of loss for the B and P frames.

We give now a brief outline of how the large-deviation approximation applies to our scheme. First, for each video in the server we construct a modified trace $y_n(j)$, $j = 1, \ldots, J$, with $y_n(j) = x_n(j)$ if the nth frame is an I frame and $y_n(j) = 0$ if the nth frame is either a P or B frame. We then calculate the frame size distribution for the modified trace:

$$\pi_j(l) := \frac{1}{N} \sum_{n=1}^{N} 1(y_n(j) = l),$$

The logarithmic moment generating function of the aggregate amount of traffic in a frame for the modified trace is:

$$\mu_Y(s) := \ln E[e^{sY}],$$

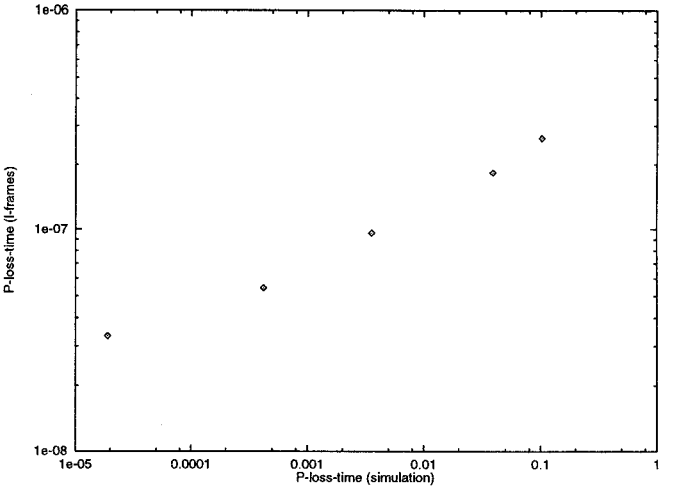

Figure 6: $P_{\text{loss}}^{\text{time}}(I)$ as a function of $P_{\text{loss}}^{\text{time}}$ for 16 KByte prefetch buffer.

where

$$Y = \sum_{j=1}^{J} Y(j)$$

and $Y(j)$ is distributed according to $\pi_j(\cdot)$. The probability that at least one I frame is lost in an arbitrary frame period is approximated as

$$\begin{aligned} P_{\text{loss}}^{\text{time}}(I) &= P(\text{"all I frames cannot be transmited in a frame period"}) \\ &\approx \frac{1}{s^* \sqrt{2\pi \mu_Y''(s^*)}} e^{-s^* R/F + \mu_Y(s^*)}, \end{aligned} \tag{6}$$

where s^* satisfies

$$\mu_X'(s^*) = R/F.$$

The admission control test is thus satisfied if

$$\frac{1}{s^* \sqrt{2\pi \mu_Y''(s^*)}} e^{-s^* R/F + \mu_Y(s^*)} \leq \epsilon,$$

where ϵ is some minute number such as 10^{-9}.

Figure 6 shows the relationship between the $P_{\text{loss}}^{\text{time}}$ (the fraction of frame periods during which loss occurs for I,P, or B frames) and $P_{\text{loss}}^{\text{time}}(I)$ (the conservative estimate for the fraction of frame periods during which loss of an I frame occurs when using selective discard). In this figure with have used a 16 KByte prefetch buffer at each of the clients. The series of points was obtained by varying the link utilization. (For visual simplicity we ignore the confidence intervals for $P_{\text{loss}}^{\text{time}}$ and only report the mid-point in the figure.) From this curve we

see, for example, that when $P_{\text{loss}}^{\text{time}}$ is 2×10^{-5} then the fraction of frame periods during which I frames are lost is 3×10^{-8}. The values of $P_{\text{loss}}^{\text{time}}(I)$ reported in this figure come from the large-deviation approximation. This approximation is conservative, as it assumes that the prefetch buffers are zero; the actual loss probability for I frames is much less. In fact, for the last point (as well as the other points) on the graph, corresponding to $P_{\text{loss}}^{\text{time}} \approx 0.1$ and 99.5% utilization, we observed absolutely no loss of I frames in the simulation!

References

[1] E. Knightly, D. Wrege, J. Liebeherr, and H. Zhang. Fundamental limits and trafeoffs of providing deterministic guarantees to VBR video traffic. In *ACM SIGMETRICS*, 1995.

[2] 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.

[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. Salehi, Z.-L. Zhang, Kurose J, and D. Towsley. Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing. In *Proceedings of ACM SIGMETRICS*, May 1996. Philadelphia, PA.

[5] 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.

[6] J. M. McManus and K. W. Ross. Video on demand over ATM: Constant-rate transmission and transport. *IEEE JSAC*, 14(6):1087–1098, August 1996.

[7] S. Shenker and C. Partridge. Specification of guaranteed quality of service. Technical report. Internet Draft; December 1995.

[8] M. Grossglauser, S. Keshav, and D. Tse. RCBR: A simple and efficient service for multiple timescale traffic. In *ACM SIGCOMM*, 1995.

[9] J.M. McManus and K.W. Ross. Prerecorded VBR sources in ATM networks: Piecewise-constant rate transmission and transport. In *to appear in Proceedings of SPIE*, Dallas, TX, 1997. Available at http://www.seas.upenn.edu/~ross/.

[10] 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.

[11] W. Feng and J. Rexford. A comparison of bandwidth smoothing techiniques for the transmission of prerecorded compressed video. In *Proceedings of IEEE Infocom*, Kobe, Japan, April 1997.

[12] 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.

[13] 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*, 15(6):1148–1166, August 1997.

[14] O. Rose. Statistical properties of MPEG video traffic and their impact on traffic modelling in ATM systems. Technical Report 101, University of Wuerzburg, Insitute of Computer Science, Am Hubland, 97074 Wuerzburg, Germany, February 1995.
ftp address and directory of the used video traces: ftp-info3.informatik.uni-wuerzburg.de /pub/MPEG/.

[15] M.I. Reiman. Some diffusion approximations with state space collaspe. In F. Baccelli and G. Fayolle, editors, *Lecture Notes in Control and Informational Sciences 60*, pages 209–240. Springer-Verlag, 1983.

[16] 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.

[17] Keith W. Ross. *Multiservice Loss Models for Broadband Telecommunication Networks.* Springer–Verlag, 1995.

[18] M. Reisslein and K. W. Ross. A join–the–shortest–queue prefetching protocol for VBR video on demand. Technical report, University of Pennsylvania, Department of Systems Engineering, January 1997. Available at http://www.seas.upenn.edu/~reisslei/.