Periodic Broadcasting with VBR-Encoded Video [*]

Despina Saparilla[1]
Dept. of Systems Engineering
University of Pennsylvania
Philadelphia, PA 19104
saparill@eurecom.fr

Keith W. Ross
Institut Eurecom
06904, Sophia Antipolis
France
ross@eurecom.fr

Martin Reisslein
GMD FOKUS
Berlin, Germany
reisslei@fokus.gmd.de

July 1998

Abstract

We consider designing near video on demand (VoD) systems that minimize start-up latency while maintaining high image quality. Recently several research teams have developed periodic broadcasting techniques for near VoD that use non-uniform segmentation. These techniques give significant reductions in start-up latency as compared with more conventional uniform segmentation. All of these schemes assume, however, that the videos are CBR-encoded. Because a CBR-encoded video has a larger average rate than an open-loop VBR encoding of the same video with the same image quality, there is potential to obtain further performance improvements by using VBR video. In this paper we develop a series of multiplexing schemes for the periodic broadcasting of VBR-encoded video. Our multiplexing schemes are based on smoothing, server buffering and client prefetching. There are two key but conflicting performance measures when using VBR video: latency and packet loss. By introducing small additional delays in our multiplexing schemes, our traced-based numerical work shows that the schemes can achieve nearly 100% link utilization with negligible packet loss. Our numerical examples show that even when the ratio of the CBR rate to the VBR average rate is a modest 1.8, start-up latency can be reduced by a factor of four or more for common scenarios.

1 Introduction

True Video on Demand (VoD) services permit subscribers to schedule an arbitrary starting time for a video of their choice. With true VoD, clients can select a video from a large number of video files, which are stored on central video servers. Requested videos are transmitted to a large population of clients through a network (e.g., cable, ADSL, or a LAN), and a distinct stream is dedicated to each user. True VoD is sometimes referred to as user centered since server and network bandwidth are strictly divided among the system's users [1] [17]. As the number of users increases, server and network bandwidth is quickly depleted. Consequently, true VoD is often considered inefficient and too costly to offer as a service.

[*] Supported partially by NSF grant NCR97-06495.
[1] Corresponding Author: currently visiting Institut Eurecom, 06904 Sophia Antipolis, France.

To provide scalable VoD, various techniques that are based on a data centered approach have been developed. In these techniques the server divides its bandwidth among distinct video objects, rather than among distinct users, and each video file is broadcast to the receivers. Broadcasting allows many clients to share a single server stream and, thus, achieves efficient utilization of both network bandwidth and server capacity [2] [15]. Techniques in which many clients share a common server stream for receiving a video object provide **near VoD**. With near VoD, the users are not offered immediate playback of the video, but instead experience a delay of the order of seconds to tens of minutes before the commencement of the video of their choice. In some techniques for providing near VoD, this start-up latency is due to a delay at the server during which requests for the same object are batched and served together using a single server stream [4]. In another set of near VoD techniques, the server periodically broadcasts each video object at fixed time intervals and clients must wait until the beginning of the broadcast session before viewing the video of their choice. Techniques of this latter type are referred to as periodic broadcasting schemes [1] [7] [6] [17].

When the number of users is very large, periodic broadcasting can be an efficient means to distribute stored video. Periodic broadcasting scales nicely, as the start-up latency is completely independent of the number of clients that wish to view the videos. The start-up latency depends, however, on the particular periodic broadcasting scheme as well as the number of videos that that are broadcasted. In broad terms, the fewer the number of videos, the greater the number of copies of each video that can be broadcasted, and the lower the initial start-up latency. Fortunately, for movies on demand, a large fraction of the demand is typically for the 10-20 most popular movies.

One simple periodic broadcasting scheme is to broadcast multiple entire copies of each video, with a new copy broadcasted every fixed interval of time (e.g., a new copy of “Star Wars” broadcasted every 20 minutes) [4]. We refer to such schemes as *periodic broadcasting with uniform segmentation*. In such a scheme, the maximum start-up latency experienced by a user is equal to the length of the video divided by the number of copies broadcasted. This latency can be long for full-length MPEG-2 encoded movies sent over channels on the order of 100 Mbps. For example, when ten movies, each encoded at 3 Mbps and each two-hours-long, are broadcasted over a 100 Mbps channel, the maximum start-up latency is 40 minutes. Beginning with the seminal paper [17], a number of non-uniform segmentation schemes have recently been proposed [1] [7] [6] [17]. Loosely speaking, these schemes reduce the initial start-up latency by broadcasting the earlier portions of the video more frequently and the latter portions less frequently.

All of the existing work on near VoD systems with periodic broadcasting is based on the assumption that the videos are Constant Bit Rate (CBR) encoded [1] [7] [6] [17]. The CBR encoding technique modifies the quantization scale during compression, which causes quality degradation in the encoded video. (With CBR encoding, the bit-rate of resulting encoded video actually fluctuates around the target CBR rate; but the video can be transmitted at the CBR rate and a small smoothing buffer at the client ensures continuity [9] [3].) For open-loop VBR encoding, the quantization scale remains constant throughout the encoding process, which often produces *highly* variable bit rates. Digital video distribution systems using satellite and cable have avoided using VBR video due its burstiness. Nevertheless, for a given movie or sporting event and *for the same quality level,* the average bit rate for CBR video is typically 2 times or more the average bit rate of VBR video [3] [16]. Therefore with VBR video there is potential for increased system efficiency.

Although non-uniform segmentation can greatly reduce start-up latency, for many practical circumstances the start-up latency remains unacceptably high for CBR-encoded video. When a near VoD system broadcasts full-length MPEG-2 encoded movies over channels on the order of 100 Mbps, the initial latencies can be large. For example, when ten movies, each encoded at 3 Mbps and each two-hours-long, are broadcasted over a 100 Mbps channel, the maximum initial start-up latency is more than 17 minutes. In this paper we develop non-uniform segmentation schemes with VBR-encoded video that significantly reduce the initial start-up latency without appreciably degrading image quality. In particular, for situations of practical interest, as the one described above, the start up latency can be reduced by a factor of 4 or more when the CBR/VBR average bit-rate ratio is a modest 1.8.

In order to obtain dramatic reductions in start-up latency with VBR-encoded video, we must allow for some small fraction of packet loss (due to link buffer overflow). The loss, however, will not be noticeable if it is extremely rare. Therefore, the challenge is develop a near VoD scheme that uses VBR-encoded video and yet has low packet loss, on the order of 10E-6 or less. (Such losses can be effectively hidden by the use of error concealment techniques such as those discussed in [10]).

In this paper we first show how non-uniform segmentation can be combined with bufferless statistical multiplexing [11] in order to create a near VoD scheme using VBR-encoded video. We present a methodology that explores the trade-off between start-up latency and packet loss. The bufferless multiplexing scheme does not provide sufficiently low loss probabilities, but it does set the stage for a series of more sophisticated schemes which increasingly offer higher performance. The first of these schemes combines smoothing with bufferless multiplexing, providing significant performance gains. The second scheme uses server-buffer multiplexing, further increasing performance. The third scheme uses client prefetching [12][13], and leads to yet further improvements. We provide extensive numerical examples that show that smoothing, server buffering, and prefetching can lead to dramatic reductions in initial start-up latency while keeping the loss probability negligible.

This paper is organized as follows. We end this section with a brief overview of non-uniform segmentation schemes for CBR-encoded video. In Section 2 we describe the general technique for providing near VoD with VBR-encoded video. We develop a methodology for determining start-up latency and loss probability. In Section 3 we study a specific segmentation scheme and present results from trace-driven simulations for bufferless multiplexing. In Section 4 we propose a series of techniques – smoothing, server buffering, and prefetching – which aim at improving performance, and we present results from a trace-driven simulation for each technique. In Section 5 we show that our VBR multiplexing schemes can dramatically reduce the CBR start-up latency.

1.1 Overview of Non-Uniform Segmentation Techniques

The first non-uniform segmentation technique referred to as *Pyramid Broadcasting* [17] divides each video into K segments of geometrically increasing sizes. The server capacity is divided evenly into K logical channels, and the i^{th} segments of all videos are broadcasted sequentially on the i^{th} logical channel. Since the first segments of the videos are the smallest in size, they are also broadcasted the most frequently. Consequently, the start-up latency for any video is significantly reduced. The reduced start-up latency achieved in [17] requires large storage buffers and high disk bandwidth at the receivers (due to the high broadcast rate of each channel). To address these two issues, a technique proposed in [1] repartitions each logical channel such that a separate sub-channel is allocated to each of the K video segments. In addition, [1] allows for p copies of each segment to be transmitted, each beginning at every short interval of time. The new technique referred to as *Permutation-Based Pyramid Broadcasting* significantly reduces disk bandwidth requirements. To substantially reduce storage requirements at the receiving end and still maintain short start-up latency, the *Skyscraper Broadcasting* technique [7] employs a different video segmentation scheme and a new broadcasting strategy. In this latter technique, each segment is broadcast at its consumption rate on a separate logical channel. Finally, the authors of [6] propose yet another segmentation scheme, which results in further reduction in start-up latency relative to [7], without compromising storage and disk bandwidth costs. This latter technique exploits receiver resources by allowing the clients to download from many subchannels simultaneously (pipelining). The use of extensive pipelining permits video division into fast (geometrically) growing video segments that are broadcast on separate channels at their consumption rate.

2 Near VoD with VBR-Encoded Video

We now present the key components of the general periodic broadcasting technique for VBR encoded video. Let *M* be the number of encoded videos to be broadcasted and let *N(m)* be the number of frames in the m^{th} video. All videos are open-loop VBR encoded. In order to keep the presentation simple, we assume that each video has a frame rate of *F* frames per second. The trace sequence of each prerecorded video is fully known; let $x_n(m)$, $n=1,\ldots,N(m)$, $m=1,\ldots,M$ indicate the number of bits in the n^{th} encoded frame of the m^{th} video. Finally, we denote the shared bandwidth between server and clients by *C* Mbps. All video streams sent by the server share the *C* Mbps. (The shared channel could be a cable or a digital satellite channel, for example).

Our basic periodic broadcasting scheme for VBR traffic works as follows. Each video is divided into *K* segments prior to broadcasting. The server broadcasts *MK* simultaneous video streams, each of which repeatedly sends a single segment of a video. Frames from the *MK* streams are statistically multiplexed into the broadcast channel without buffering. Bits are lost whenever the broadcast rate exceeds the channel rate. The server broadcasts each video stream at rate F frames per second, the consumption rate of the videos. A client that wishes to see a particular video tunes to the stream that is repeatedly broadcasting the first segment of that video. The user then waits until the beginning of the segment starts to arrive. We refer to the maximum delay experienced by the user as the *start-up latency*. At the next broadcast of the first segment the client begins to receive and concurrently display frames from the beginning of the segment. As with the CBR schemes, the client downloads the remaining segments of the video according to a specific download strategy [6][7]. The choice of download strategy depends on the ability of the client to employ pipelining, i.e., on its ability to receive frames from a number of video streams simultaneously. The download strategy is specified by *q*, the number of simultaneous streams from which the client can download frames at any time. For example, for $q = 4$, the client downloads segments at their next occurrence for at most four streams at a time.

A central characteristic of a periodic broadcasting scheme (CBR and VBR) is the manner in which the videos are segmented. In general, each video is divided into *K* segments according to a series of terms referred to as broadcast series [1][7][6][17]. Let $[e_1, e_2, \ldots, e_{K-1}, e_K]$ be a general broadcast series. The series specifies that the first segment of each video consists of e_1 segmentation units, the second segment of e_2 segmentation units, etc. Without any loss of generality, set $e_1 = 1$. Let $N_i(m)$ indicate the number of frames in the i^{th} segment of the m^{th} video The broadcast series implies that successive segment sizes are related by:

$$N_i(m) = e_i N_1(m), \qquad i = 2, \ldots, K. \tag{1}$$

The size of the first video segment is determined by the equation

$$N_1(m) = \frac{N(m)}{(e_1 + e_2 + \cdots + e_{K-1} + e_K)}. \tag{2}$$

Thus, a broadcast series $[e_1, e_2, \ldots, e_{K-1}, e_K]$ and video length *N(m)* completely specifies all segment sizes.

An important requirement of a periodic broadcasting scheme is that it must allow the delivery of the video in a continuous and timely fashion. In other words, the delivery scheme must permit the display of the decoded video at the client without interruptions. This requirement is referred to as the *continuity condition*. Whether a certain scheme satisfies the continuity condition, given the video consumption rate *F* and the video size *N(m)*, depends on the value of *q* and the specific form of broadcast series used [6]. For example, consider *q = 1*, in which case the client can download frames from only one stream at a time. In this case, the uniform broadcast series $[1, 1, \cdots, 1]$ is the only type of series that results in a feasible delivery scheme, i.e., one series satisfies the continuity condition. When *q = 2*, a number of different broadcast series satisfy the continuity condition. For example, the series of increasing terms given in [7] and [6] both satisfy the continuity requirement for *q = 2*. Finally, in the extreme case when $q = K$ a larger set of broadcast series results in feasible delivery schemes. For example the fast growing geometric series $[1, 2, \ldots, 2^{K-1}]$ is one such series that meets the continuity condition.

As a specific example, consider the continuity condition for $q = K$ and the geometric broadcast series $[1, 2, \ldots, 2^{K-1}]$. The broadcasting strategy for this series is illustrated in Figure 1 for *K = 4* and just a single video. Since *q = K*, the client can download frames from all video streams simultaneously. As a result, each of the four segments can be received at its next broadcast. The following argument shows that the continuity condition is indeed satisfied for this broadcast series and *q* combination. Consider two successive video segments of sizes $N_i(m)$ and $N_{i+1}(m)$. The continuity condition is satisfied if the second segment becomes available before or at the time the broadcast of the previous segment ends. Since the sizes of the two segments are related by equation $N_{i+1}(m) = 2 \cdot N_i(m)$, the broadcasts of the segments either begin or end at the same time. In the case when the broadcasts begin simultaneously, segment *i+1* becomes available early and can be downloaded and stored by the client in the playback buffer. In the case when the broadcasts end at the same time, a broadcast of segment *i+1* immediately follows that of segment *i*, and continuity is maintained.

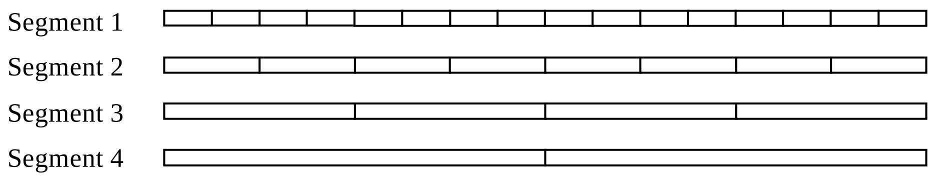

Figure 1: Broadcasting strategy for geometric series with $e_k=2^{k-1}$.

Start-up latency is defined as the maximum delay experienced by a user before the commencement of a video. This latency is equal to the maximum access time of the first segment of the video, which in turn is equal to the broadcast duration of the segment. We let *L(m)* indicate the start-up latency for the *m*th video. We have:

$$L(m) = \frac{N_1(m)}{F}. \tag{3}$$

For a general broadcast series $[e_1, e_2, \ldots, e_{K-1}, e_K]$ where $e_1 = 1$, the start-up latency is given by:

$$L(m) = \frac{N(m)}{F \cdot \sum_{i=1}^{K} e_i}. \tag{4}$$

As seen from (4), the start-up latency associated with a periodic broadcasting scheme is decreased for higher values of *K*. Additionally, for a given value of *K*, the start-up latency is decreased when a fast growing broadcast series is utilized. This is due to the fact that the sum of a series of fast growing terms is larger than that of a series of slow growing terms.

With VBR video, there are two performance measures, start-up latency and loss probability. As we shall see, there is a trade-off between these two measures. We define the probability of loss to be the long-run fraction of bits lost from the video streams during broadcasting. To determine this fraction, we index each video stream by a tuple *(m,k)*, where *m* indicates the video and *k* the specific video segment that is sent by the stream. Loss of bits occurs when the aggregate bit rate of the traffic (i.e., from all *MK* streams) exceeds the link's capacity, *C*. Let $y_t(m,k)$ denote the number of bits sent by stream *(m,k)* during frame time *t*. Then, $y_t(m,k)$ can be expressed as a function of the trace sequence $x_m(n)$ as follows:

$$y_t(m,k) = x_m(j), \tag{5}$$

where *j* is given by

$$j = \sum_{i=1}^{k-1} N_i(m) + remainder\left(\frac{t}{N_k(m)}\right). \tag{6}$$

Note that *j* represents the index for the frame of the *m*th video (i.e., *j=1,..,N*) that is sent during frame time *t*. Observe also that the value of *j* depends on the resulting segment sizes after division of the video. We next determine y_t, the total number of bits that reach the link during frame time *t*. Given $y_t(m,k)$ for *m=1,…,M*, *k=1,…,K*, y_t is computed as follows:

$$y_t = \sum_{m=1}^{M} \sum_{k=1}^{K} y_t(m,k)\,. \tag{7}$$

In our bufferless model, bits are lost from the video streams if the aggregate amount of traffic that arrives at the link during frame time t exceeds the link's capacity. Thus, loss occurs in frame time *t* if

$$y_t > \frac{C}{F}\,. \tag{8}$$

We express the long-run fraction of traffic lost by P_{loss}. Thus, we have:

$$\begin{aligned} P_{loss} &= \lim_{T\to\infty} \frac{number\ \ of\ \ bits\ \ lost\ \ up\ through\ \ frame\ \ time\ \ T}{total\ \ number\ \ of\ \ bits\ \ sent\ \ up\ \ through\ \ frame\ \ time\ \ T} \\ &= \lim_{T\to\infty} \frac{\sum_{t=1}^{T} (y_t - C/F)^+}{\sum_{t=1}^{T} y_t}. \end{aligned} \tag{9}$$

Small start-up latency and small loss probability are conflicting objectives. The start-up latency is minimized for high values of *K*. On the other hand, the aggregate amount of traffic that reaches the link in a frame time increases with *K* in approximately a linear fashion. Thus, the fraction of bits lost in the long-run also increases with *K*. In the next section, we present numerical results from the simulation of a specific periodic broadcasting scheme that illustrate the tradeoff between start-up latency and loss probability.

3 Numerical Example: Geometric Series

In order to illustrate the use of VBR video with periodic broadcasting, we focus our numerical work on one download strategy and broadcast series. The techniques developed in this paper can be applied to an arbitrary download strategy and broadcast series (as long as the continuity condition holds).

Specifically, we use $q = K$ and the geometric series $[1, 2, \ldots, 2^{K-1}]$ to segment the videos. As a consequence, the client can download each of the K segments at their next occurrence. Recall from Section 2 that a periodic broadcasting scheme utilizing the above broadcast series and pipelining combination satisfies the continuity condition. In our numerical example, we also make the assumption that receiver storage is not a constraining factor. In other words, we assume that playback buffers at the clients are large enough to receive and store all incoming segments without loss. When receiver storage is a constraint, however, an approach presented in [7], in which segment sizes are restricted to a maximum value *W*, can instead be employed.

We obtained 7 MPEG encoded movies from the public domain [5][8][14]. The trace of each movie gives the number of bits in each frame. In the simulation study, the seven encoded movies were used to create 10 "pseudo traces" each 160,000 frames long. Table 1 summarizes statistics associated with the resulting traces. The 10 traces used in the numerical study were created from the 7 movies in the following manner. Five traces were created using the encoded movies in [14]. Since each of the original movies is 40,000 frames long, to create each of the first 5 traces shown in Table 1, the trace sequence of each movie was repeated four times. Each resulting trace sequence of 160,000 frames was then multiplied by a constant to bring the average bit rate to 2 Mbps. The sixth trace in Table 1 was created by repeating four times the first 40,000 frames of the MPEG encoding obtained from [8]. Note that the resulting trace was again manipulated such that its average bit rate is 2 Mbps. Finally, the MPEG encoding obtained from [5] was first divided into four parts 40,000 frames each. The resulting movie segments were repeated four times to create 4 different trace sequences of 160,000 frames. The trace sequences were finally multiplied by the appropriate constants to create the last four traces illustrated in Table 1 with average bit rates equal to 2 Mbps. Although the ten pseudo traces are not traces of actual movies, we believe that they reflect the characteristics of MPEG-2 encoded movies (highly bursty, long-range scene dependence, average rate about 2 Mbps).

	Frames		GoPs	
Trace	*Peak/ Mean*	*St. Dev bits*	*Peak/ Mean*	*St. Dev bits*
bond	10.1	2,114,056	4.2	1,104,733
lamps	18.4	3,062,443	3.3	1,570,752
mr. Bean	13	2,339,592	5.0	481,632
soccer	6.9	1,914,516	3.7	1,872,193
terminator	7.3	1,863,047	2.8	2,133,982
wiz.of oz	8.4	2,485,807	3.2	3,482,278
star wars 1	10.9	2,447,206	3.7	2,367,985
star wars 2	13.2	2,341,372	4.4	2,571,459
star wars 3	12	2,312,285	3.1	2,765,630
star wars 4	8.5	2,138,637	3.2	2,960,508

Table 1: Trace Statistics

In summary, we have $M = 10$ videos each of which has 160,000 frames, i.e., $N(m) = N = 160{,}000$ frames. Each video has a length of approximately 107 minutes. All videos are VBR encoded with $F = 25$ frames per second. As illustrated in Table 1, the traces used in the numerical study have high peak/mean ratios and standard deviations. Finally, to determine the effect of server bandwidth on performance, we allowed the link capacity C to vary between 85 and 205 Mbps.

Our numerical study of the periodic broadcasting scheme focuses on two performance measures: start-up latency and probability of loss. The start-up latency associated with the scheme is computed according to equation (4) for a general periodic broadcasting scheme. By using the sum of a geometric series with $e_k = 2^{k-1}$ as the sum of the generic broadcast series, we have the latency

$$L = \frac{N}{F \cdot (2^K - 1)} \tag{10}$$

for each video. The probability of loss is expressed by the expected fraction of bits lost from the video streams during broadcasting. In general, to obtain an accurate approximation of P_{loss}, it becomes necessary to perform a simulation of the periodic broadcasting scheme during a large number of frame times. In this specific case, however, the use of geometric series [1, 2,…, 2^{K-1}] for segmenting the videos introduces a periodicity in the aggregate traffic pattern y_t. The same aggregate traffic pattern repeats every N_K frame times where N_K is the size of the largest segment of each video. As a result, we can determine P_{loss} by simulating the system for the first N_K frame times, and collecting statistics regarding the number of bits lost in each frame time. Thus, we compute the loss probability P_{loss} as follows:

$$P_{loss} = \frac{\sum_{t=1}^{N_K} (y_t - \frac{C}{F})^+}{\sum_{t=1}^{N_K} y_t} . \tag{11}$$

To obtain confidence intervals on the loss probability associated with the periodic broadcasting scheme, we must perform multiple independent replications of the broadcasting in which the broadcasted traffic pattern varies. To allow for these multiple replications using the available set of traces, we introduce a random shift in what we consider the starting point of each trace. Thus, in each replication of the broadcasting scheme, 10 random numbers are drawn from a uniform distribution between [1, 160000] to determine random starting points for each of the traces. Starting from these random points, a perturbed trace of 160,000 frames is obtained for each of the videos by wrapping each trace around until the original starting point is reached. In each replication, the segmentation of the videos is performed in the manner specified by the broadcast series using new perturbed traces.

3.1 Bufferless Statistical Multiplexing

In this subsection, we study the performance of the periodic broadcasting scheme when the original VBR traffic is statistically multiplexed over a bufferless link. The results illustrate the start-up latency and probability of loss levels achieved by different values of *K*. We only consider values of *K* that generate start-up latencies in the range 0-16 minutes and loss probabilities below 0.1. The start-up latencies resulting from different *K* values are independent of the link's capacity. Recall that when the geometric series is used to segment the videos, start-up latency decreases exponentially with *K*. It is easily seen from (10) that for start-up latencies below 2 minutes, *K* must be at least 6. Start-up latencies below half a minute result from values of *K* that are at least 9. As the number of segments *K* increases, the probability that bits will be lost from the video streams also becomes higher. Our results specify this tradeoff between start-up latency and the probability of loss.

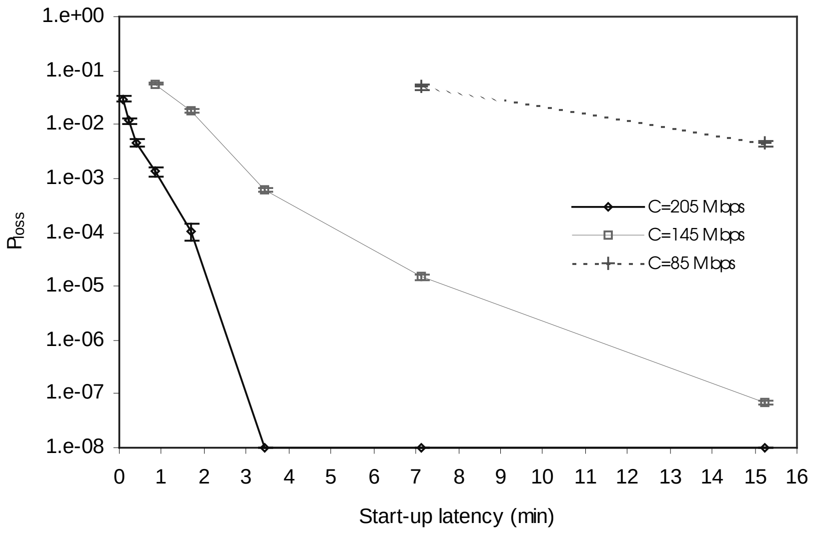

Figure 2: Bufferless statistical multiplexing.

In Figure 2, the probability of loss, P_{loss}, is plotted against start-up latency for different link capacities. The figure shows the average loss probability and 90% confidence intervals for the average, whose length is smaller than 10% of the estimated mean. As the figure illustrates, the loss probability associated with each start-up latency value, or equivalently with each value of *K*, varies according to link capacity. Each point on the curve for a single bandwidth level corresponds to a specific value of *K*. Depending on the available link capacity, feasible *K* values range between 3 and 10. Lower *K* values result in start-up latencies that exceed 16 minutes while higher values generate loss probabilities above 0.1. Figure 2 illustrates that there exists a tradeoff between P_{loss} and start-up latency with varying K. As the value of K increases, start-up latency decreases while P_{loss} increases. For start-up latency values below 2 minutes, the probability of loss is high (i.e., in the order of 10^{-4} or higher) for all the levels of link bandwidth examined. While this loss-latency tradeoff is a key characteristic of periodic broadcasting with VBR encoded video, it is not an issue for broadcasting CBR encoded video. The constant rate traffic in the CBR schemes allows the channel's bandwidth to be allocated among the videos in a manner that guarantees no loss due to channel overflow.

The results obtained for bufferless statistical multiplexing of the VBR encoded streams indicate that if it is desirable to achieve latency values below 2 minutes, high probabilities of loss will be incurred resulting in unacceptable levels of quality degradation in the decoded video. This has motivated us to refine our multiplexing schemes in order to improve performance. In the following sections, we examine three different methods for limiting loss. The first is the implementation of GoP smoothing on the VBR traces prior to broadcasting. The second is buffered statistical multiplexing of the video streams through the addition of a finite buffer at the server link. The third uses prefetching of video frames during periods of time when the shared link's bandwidth is under utilized.

4 Improving Performance

4.1 GoP Smoothing

In this subsection we investigate the effect of GoP smoothing on the performance of the broadcasting scheme. We first obtain results for the case when the video traces are smoothed over each GoP period. We again examine the start-up latency and the resulting probability of loss for link capacities ranging from 85 to 205 Mbps. The results of this study are plotted in Figure 3. The total start-up latency shown in the plot is the sum of the maximum access time for the first video segment and the delay introduced due to smoothing over one GoP period. This additional delay is equal to the length of a GoP period. For example, when the GoP size is equal to 12 frames and the broadcast rate is 25 frames per second, the additional start-up delay introduced due to smoothing over one GoP period is equal to 12/25 seconds or 0.48 seconds.

We observe a significant improvement in the loss probability due to GoP smoothing. The decrease achieved in the probability of loss associated with a specific K is substantial for all three link capacities studied. Note that this improvement in P_{loss} occurs at the expense of only a small increase in the total playback delay, i.e., an additional delay of 0.48 seconds.

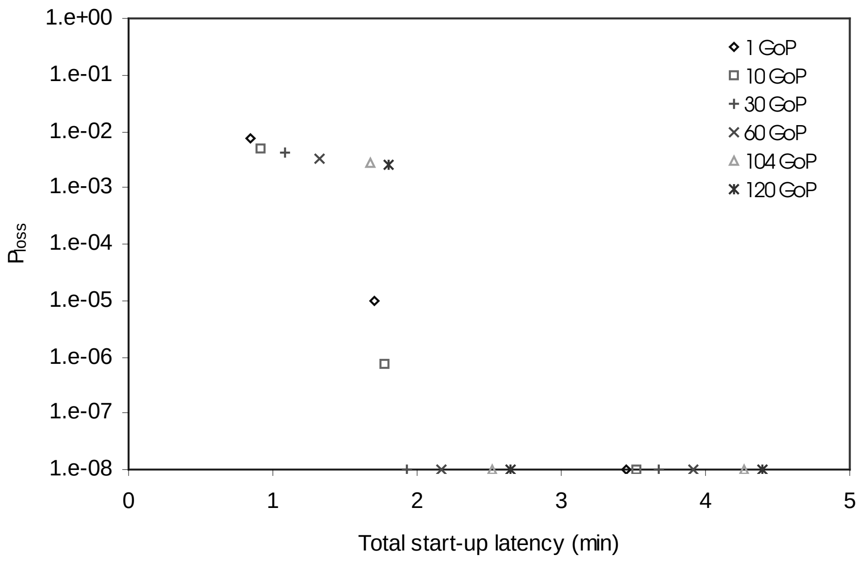

Figure 3: Smoothing over many GoP periods (C=145 Mbps).

We now focus on the case when the link capacity is 145 Mbps. Our numerical study aims at examining the effect of further smoothing on the loss probability. Instead of smoothing the video traces over each GoP period, we also employ smoothing of each trace over intervals that consist of a larger number of GoP periods. For instance, we smooth over intervals consisting of 10, 30, etc., GoP periods. We compare the effect of these different smoothing policies on loss and start-up latency. The results are shown in Figure 4. We concentrate on values of K equal to 6 and 7 since for smaller K P_{loss} is zero for all smoothing policies. Let's first consider the cluster of points in the 1.5-3 minute interval on the latency scale. These points correspond to different smoothing policies for the case when K is equal to 6. Clearly, for $K = 6$, smoothing over intervals of 10 GoP periods, results in a considerable decrease in P_{loss}. Further improvement in P_{loss} occurs with smoothing over intervals of 30 GoP periods. In this case, P_{loss} becomes zero. Note that smoothing over even longer intervals of 60 or 120 GoP periods introduces a longer smoothing delay without affecting P_{loss}. Thus, in the case when $K = 6$, smoothing over intervals longer than 30 GoP periods is not only unnecessary but also undesirable. We refer to points that correspond to longer total start-up latencies with no further improvement in P_{loss} as **dominated**. Dominated points also exist in the case when $K = 5$ (points in the 3-5 minute interval). In this case, smoothing over 1 GoP interval is sufficient for bringing the loss probability to zero. Now consider the leftmost cluster of points in the 0.5-2 minute latency interval for which $K = 7$. These points are non-dominated in the sense there is always an improvement in P_{loss} with increasing latency. We observe however that the decrease in P_{loss} achieved by smoothing over longer periods is not significant relative to the added delay introduced by smoothing.

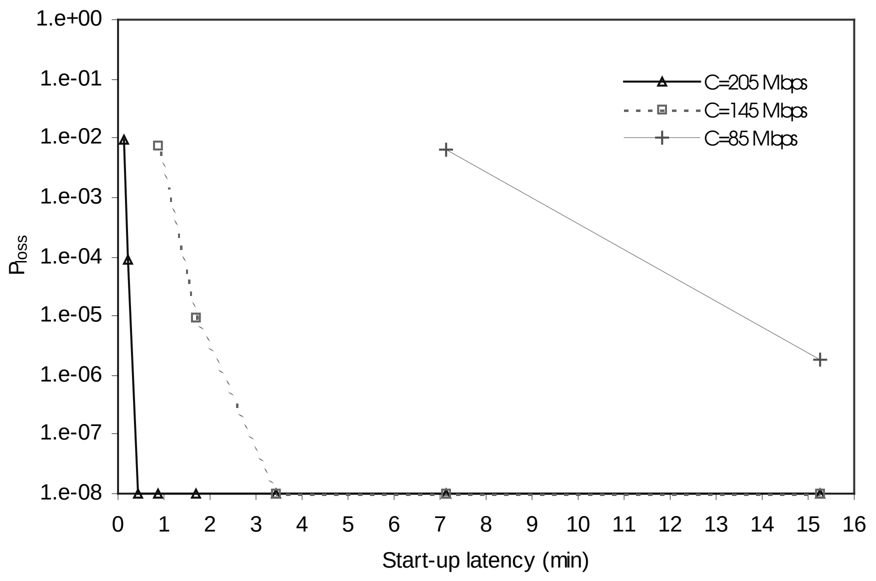

Figure 4: Bufferless multiplexing with smoothing over each GoP period.

Finally, observe that smoothing over very long intervals (i.e., intervals of 120 GoP periods), results in additional delays that are significant enough to also cause dominance between clusters of points that correspond to different *K* values. In particular, division of the video files into 7 segments when smoothing over intervals of 120 GoP periods is implemented, results in higher latency than division into 6 segments with smoothing over 1 GoP period. Since the effect of extensive smoothing is not substantial enough to achieve P_{loss} that is lower than the level achieved in the latter case (for lower *K*), the point corresponding to $K = 7$ is dominated by that corresponding to $K = 6$. In general, when different clusters of points overlap on the horizontal axis of latency, then overlapping points belonging to the cluster with the higher P_{loss} values are dominated by those belonging to the other. In conclusion, smoothing over a higher number of GoP periods does not have an adverse effect when low start-up latencies are desirable.

4.2 Buffered Statistical Multiplexing

In this section we consider buffered statistical multiplexing of the video streams by introducing a buffer of finite size at the server link. No smoothing is performed on the video traces. We once more focus on the case when link capacity is 145 Mbps. We introduce a finite buffer and vary its size in the range 72 to 8700 Mbits, which corresponds to an additional start-up delay of 0.5 to 60 seconds. The results are shown in Figure 5 for six different buffer sizes. To facilitate comparison of results, we include in the chart the case of bufferless statistical multiplexing. The total start-up latency in the buffered case is the sum of the maximum access time for receiving the first video segment plus an added delay due to the buffer. The added delay is equal to *B/C* seconds, where *B* is the buffer size, which is the maximum possible delay that can be introduced due to buffering.

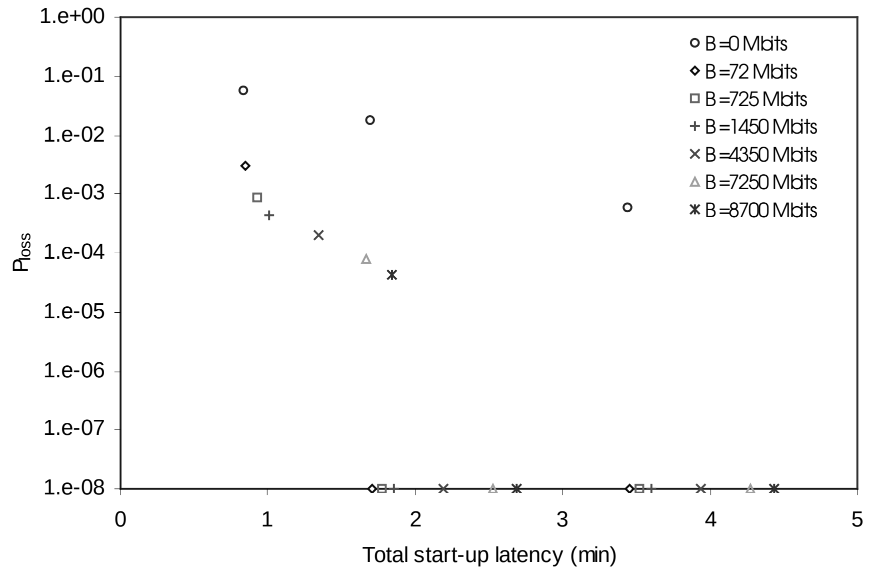

Figure 5: Buffered statistical multiplexing (C=145 Mbps).

Figure 5 shows results for values of *K* equal to 5, 6 and 7. We observe that a buffer of size 72 Mbits has a significant positive effect on the loss probability in comparison to the case of bufferless statistical multiplexing with no smoothing. For instance, when $B = 72$ Mbits, a start-up latency of approximately 1.7 minutes ($K = 6$) can be achieved with P_{loss} equal to zero. Bufferless statistical multiplexing, on the other hand, results in P_{loss} in the order of 10^{-2}. Increasing the size of the buffer to higher values than 72 Mbits, when $K = 6$, results in dominated points as it introduces longer start-up delays with no further reduction in P_{loss}. In the case when $K = 7$, increasing the size of the buffer achieves consistent improvement in P_{loss}. Note however, that the improvement in P_{loss} becomes less significant as the buffer sizes increase. The most dramatic improvement occurs for an increase of the buffer size from 0 to 72 Mbits. In this case, the value of P_{loss} decreases by an order of magnitude with only a negligible increase in the start-up latency of 0.48 seconds. Increasing the buffer size from 4350 to 8700 Mbits, however, results in a significant added delay of 0.5 minutes for a more moderate improvement in P_{loss}. As it is seen in Figure 5, using a buffer of 8700 Mbits when $K = 7$ generates a dominated point since the points corresponding to $K = 6$ and $B = 72$ or 725 Mbits achieve smaller start-up latencies and lower loss probabilities. This result indicates that in this example, utilizing buffers that introduce (maximum) delays longer than 30 seconds is not desirable. To limit loss it is instead preferable to use a smaller *K*.

4.3 Join-the-Shortest Queue Prefetching

The Join-the-Shortest-Queue prefetching protocol was originally developed for the client centered streaming of VBR encoded video over a shared bufferless link [12]. Before we discuss how the JSQ protocol can be applied to the data centered near VoD systems studied in this paper, we briefly expound on its underlying idea. The JSQ protocol is based on the observation that due to the VBR nature of the multiplexed video streams there are frequent periods of time during which the shared links' bandwidth is underutilized. During these periods the server can prefetch video frames from any of the ongoing video streams 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. The JSQ protocol also specifies the policy for selecting the prefetched frames. This policy is the Join-the-Shortest-Queue (JSQ) policy which can be roughly described as follows: within each frame period the server repeatedly selects frames from the connections that have the smallest number of prefetched frames in their client buffers. The empirical work with MPEG-1 traces in [12] indicates that prefetching combined with the JSQ policy gives dramatic reductions in loss. In particular, if each client dedicates a small buffer to the video streaming application, JSQ prefetching allows for almost 100 % utilization on the shared link with negligible loss. For a detailed discussion of the JSQ protocol for client centered video streaming we refer the interested reader to reference [12].

We now proceed to explain how to apply the JSQ protocol to the data centered near VoD system. Towards this end we introduce the concept of virtual buffers. We assign each stream *(m, k)* a virtual buffer which operates as follows: the virtual buffer of stream *(m, k)* tracks the buffer contents of a client that is tuned in to stream *(m, k)* all the time and displays segment *(m, k)* over and over again. Note that the system of *MK* virtual buffers receiving *MK* distinct video streams constitutes a client centered video streaming system. The JSQ protocol of [12] can therefore readily be applied to the system of virtual buffers. We first describe in detail the JSQ protocols' operation in the system of virtual buffers. We shall then explain how the system of virtual buffers relates to the actual near VoD system.

In explaining the details of the JSQ protocol we divide time into slots of length $1/F$. Let $p_t(m, k)$ denote the number of prefetched frames in virtual buffer *(m, k)* at the beginning of slot *t*. Let $\Delta_t(m,k)$ denote the number of frames that arrive to virtual buffer *(m, k)* during slot *t*. At the end of each slot one frame is removed and displayed provided the virtual buffer holds one or more frames. Thus

$$p_{t+1}(m, k) = [\, p_t(m, k) + \Delta_t(m,k) - 1]^+ . \tag{12}$$

If the frame that is supposed to be removed and displayed at the end of the slot does not arrive in time the virtual buffer suffers frame starvation and the frame is considered lost. The server skips the transmission of a frame that will not meet its deadline at the virtual buffer. For each of the *MK* virtual buffers the server keeps track of the buffer contents $p_t(m, k)$ through recursion (12).

During each slot of length *1/F* seconds the server must decide which frames to transmit from the *MK* ongoing streams. This is done according to the Join-the-Shortest-Queue (JSQ) prefetching policy. The maximum number of bits that can be transmitted in a slot is *C/F*. The JSQ prefetch policy attempts to balance the number of prefetched frames across all virtual buffers. In describing the policy we drop the subscript *t* from all notations. Let *z* be a variable that keeps track of the total number of bits sent within a slot; *z* is initialized to zero at the beginning of every slot. At the beginning of the each slot the server determines the stream *(m*, k*)* with the smallest *p(m, k)* and checks whether

$$z + x_{\sigma(m^*,\, k^*)}(m^*) \quad \leq \quad C / F, \tag{13}$$

where $\sigma(m^*, k^*)$ is the frame of video *m** that is being considered for transmission. If this condition holds, then we transmit the frame, increment *p(m*, k*)* and update *z*. If condition (13) is violated, we remove the stream *(m*, k*)* from consideration and find a new stream *(m*, k*)* that minimizes *p(m, k)*. If the condition (13) holds for the frame of the new stream *(m*, k*)*, we transmit the frame and update *p(m*, k*)* and *z*. We then continue the procedure of transmitting frames from the streams that minimize the *p(m, k)*'s. Whenever a frame violates condition (13) we skip the corresponding stream and find a new stream *(m*, k*)*. When we have skipped over all of the streams we set $p(m, k) = [p(m, k) \ -1]^+$ for all $m = 1,...,M$ and $k = 1,...,K$ and move on to the next slot.

Having discussed how the JSQ protocol feeds the virtual buffers and how the virtual buffers operate we now explain how to run the JSQ protocol in the actual near VoD system. In order to employ the JSQ protocol in the near VoD systems, all the server needs to do is to schedule the broadcast of the frames of the *MK* video streams as if it were sending them to the *MK* distinct virtual buffers. The clients tune in to a video as before and store the frames of the currently displayed segment that arrive ahead of time and retrieve and display them when their deadline has come. Note in particular that prefetching does not interfere with the continuity condition.

We now explain how the virtual buffers relate to the buffers in the clients of the near VoD system. Recall from the discussion on the continuity condition in Section 2 that with the geometric broadcast series with $e_k = 2^{k-1}$, the lengths of any two successive segments *(m, k)* and *(m, k+1)* satisfy the relationship $N_{k+1}(m) = 2\, N_k(m)$. As a consequence the two segments either begin or end at the same time. First, consider the case when the two segments end at the same time, that is, the end of segment *(m, k)* coincides with the end of segment *(m, k+1)*. In this case the client starts to receive the next broadcast of segment *(m, k+1)* immediately after segment *(m, k)* has ended and displays segment *(m, k+1)* right away. Note that for the duration of this broadcast of segment *(m, k+1)* the buffer contents of the client in the near VoD system and the buffer contents of the virtual buffer in the virtual buffer system are exactly the same. This implies that whenever loss due to frame starvation occurs in the virtual buffer system the near VoD system suffers exactly the same loss.

Next, we espouse the case when the segments begin at the same time. In this case the client receives and displays segment *(m, k)* while segment *(m, k+1)* is being received and stored. When segment *(m, k)* ends the beginning of segment *(m, k+1)* is retrieved from memory and displayed. In other words, the display of segment *(m, k+1)* is delayed by $N_k(m)$ frame periods by the client. Note that this implies that whenever frame starvation occurs at the virtual buffer it occurs $N_k(m)$ frame periods later at the client in the near VoD system. The long-run loss probability is therefore the same for both systems.

In Figure 6 we plot the results of a simulation study of the JSQ protocol. For this experiment we set the bandwidth to C = 145 Mbps. We plot the loss probability as a function of the start-up latency. We compare the JSQ results with the results obtained for bufferless statistical multiplexing in Section 3.1. Note that the JSQ protocol runs over a bufferless link. We observe that the JSQ protocol brings significant improvement over simply multiplexing the video stream onto the bufferless link. For K = 7 (corresponding to a start-up latency of 50.4 seconds) the loss probability drops from roughly 6 10^{-2} to

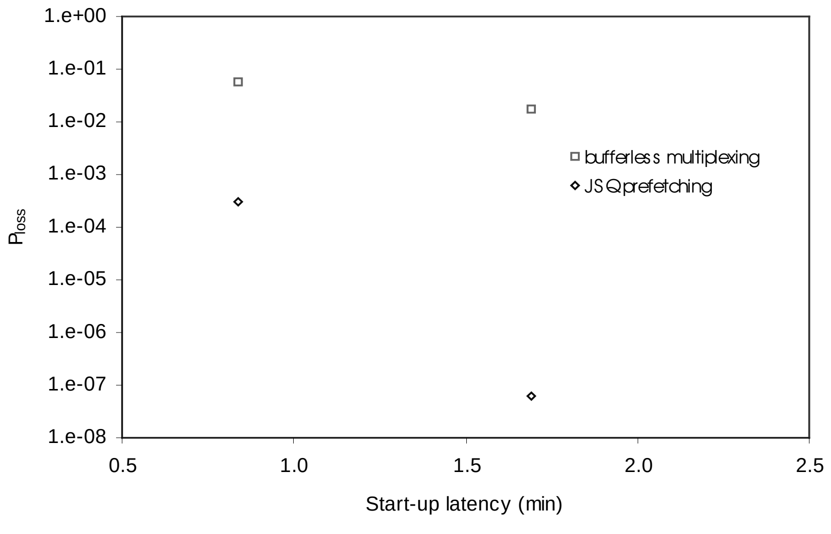

Figure 6: Comparison of JSQ prefetching and bufferless statistical multiplexing.

approximately 3 10^{-4} with JSQ prefetching.

We next explore the improvements in performance that can be achieved by a refinement of the JSQ protocol. This refinement allows the virtual buffers (and also the clients in the near VoD system) to build up a reserve of prefetched frames over a certain period of time before frames are consumed. We refer to the length of the period of time during which frames are prefetched but not consumed as the *prefetch delay*. Let d_{pre} denote the prefetch delay in frame periods. We refer to the refined JSQ protocol as *JSQ prefetching with prefetch delay*. The total start-up latency when JSQ prefetching with prefetch delay is employed is the sum of the duration of the first video segment and the prefetch delay, that is, $L = (N_1 + d_{\text{pre}})/F$.

We now show that JSQ prefetching with prefetch delay of segments generated according to the geometric broadcast series with $e_k = 2^{k-1}$ satisfies the continuity condition. The timing of prefetching with prefetch delay is illustrated in Figure 7. Consider any two successive segments *(m, k)* and *(m, k+1)*. Recall that the lengths of the segments are related by $N_{k+1}(m) = 2\ N_k(m)$ and thus the segments either begin or end at the same time. First, consider the case when the two segments begin at the same time. Since the segments begin at the same time prefetching for the segments also starts at the same time (d_{pre} frame periods before the display of segment *(m, k)* starts). The start of segment *(m, k+1)* is hence already stored in the client buffer when the display of segment *(m, k)* ends. Now, consider the case when the segments *(m, k)* and *(m, k+1)* end at the same time. In this case the server starts to prefetch frames for the next broadcast of segment *(m, k+1)* d_{pre} frame periods before the current broadcast of that segment ends. This ensures that the start of segment *(m, k+1)* is available when segment *(m, k)* ends. For a detailed

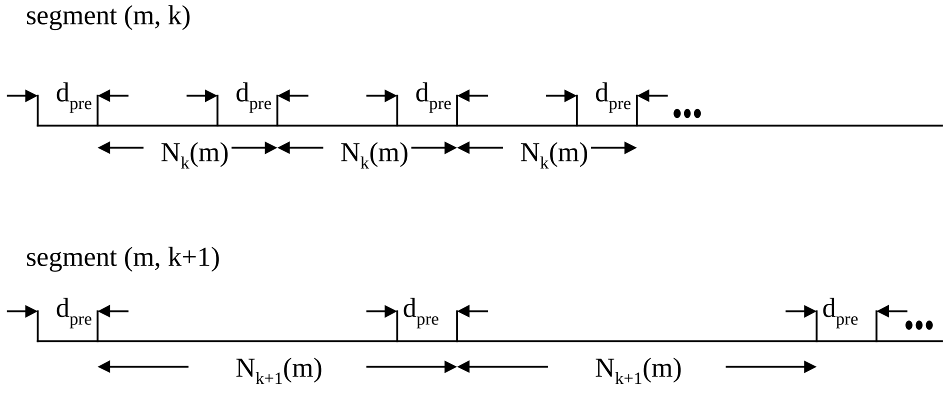

Figure 7: Timing at JSQ prefetching with prefetch delay. Prefetching starts d_{pre} frame periods before the first frame is consumed at the virtual buffer.

discussion of the implementation issues associated with the JSQ prefetching protocol, we refer the reader to the Appendix.

In Figure 8 we show the results of a simulation of JSQ prefetching with prefetch delay for the case when the link capacity is 145 Mbps and *K = 7*. We observe that the introduction of the prefetch delay improves the loss probability significantly. For a prefetch delay of 10 seconds, the loss probability decreases from $3 \cdot 10^{-4}$ to $9 \cdot 10^{-5}$. Increasing the prefetch delay to 50 seconds results in a total start-up latency of 100.4 seconds and an even lower loss probability of $6.6 \cdot 10^{-6}$. Observe from Figure 6, however, that JSQ prefetching without prefetch delay results in a total start-up latency of 100.7 seconds and a loss probability equal to $6 \cdot 10^{-8}$ for the case when *K = 6*. Our results thus indicate that to achieve a very low loss probability, it is preferable to use a smaller K rather that a long prefetch delay. This observation parallels the conclusion of Section 4.2 on buffered multiplexing which indicated that smaller K gave better performance than the use of very large buffers.

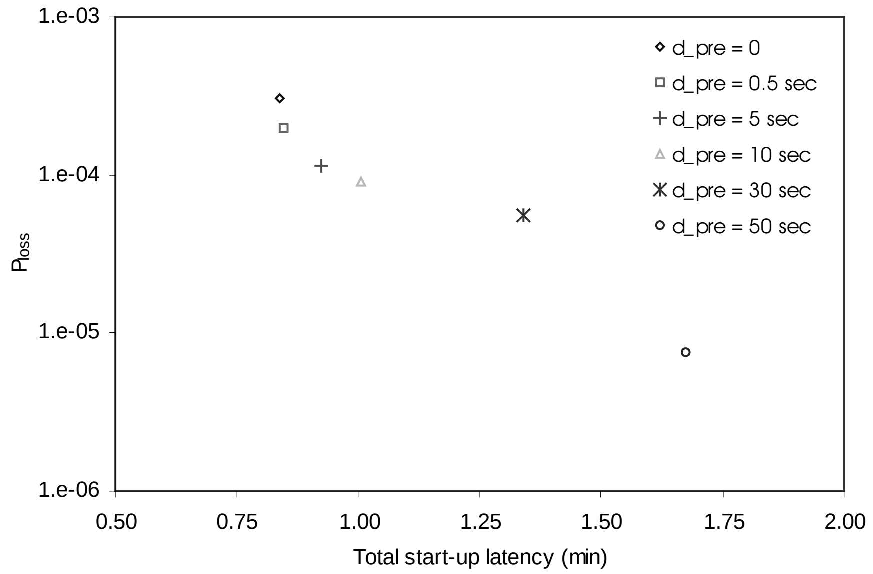

Figure 8: JSQ prefetching with prefetch delay.

We next compare the performance of buffered multiplexing, GoP smoothing and JSQ prefetching in terms of their effectiveness in limiting the loss probability. We generate the *dominance curves* corresponding to Figures 4, 5 and 8. The dominance curves specify the non-dominated points generated by each technique for different levels of the key parameters (i.e., different buffer sizes, smoothing intervals and prefetching delays). The results, illustrated in Figure 9, indicate that for similar latencies, JSQ prefetching gives the lowest loss probabilities. We observe that the loss probabilities associated with buffered multiplexing can be an order of magnitude higher than the ones generated by JSQ prefetching. We note however, that buffered multiplexing attains the performance of JSQ prefetching for extremely low levels of loss in this example. Finally, as it is clearly seen in Figure 9, JSQ prefetching and buffered multiplexing is more effective in terms of limiting the loss probability than GoP smoothing.

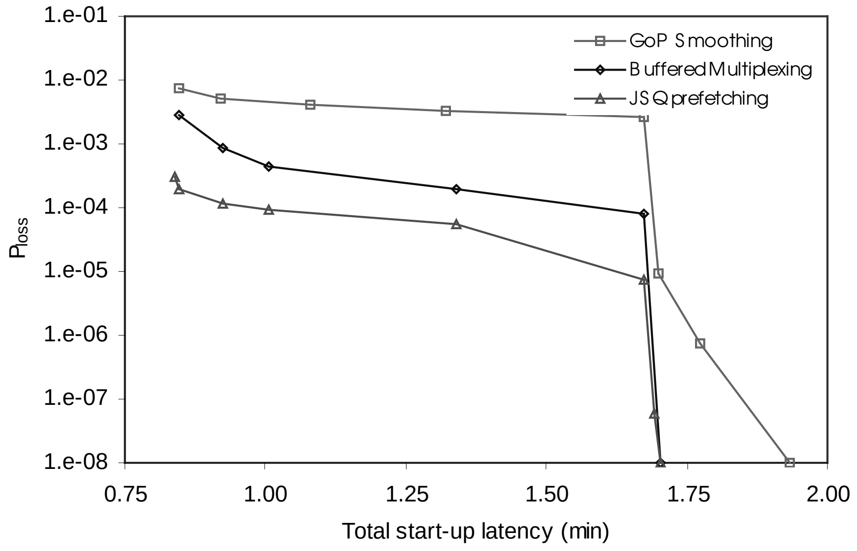

Figure 9: Dominance curves for GoP smoothing, buffered multiplexing and JSQ prefetching.

5 VBR and CBR Compared

Having shown how to design high-performance periodic broadcasting schemes for VBR-encoded video, we now compare the latency performance of CBR and VBR encoded video. In order to make a true comparison, we need to know how much CBR bandwidth is needed to achieve the image quality of open-loop VBR encoding. Unfortunately, we do not have this information for the traces in this paper. However, recent studies have shown that for movies and sporting events, the ratio of the average rate for CBR encoding to the average rate for VBR encoding is in the 2.0 range if not greater [3] [16]. Therefore, to compare VBR with CBR we will make the conservative assumption that the ratio is 1.8, i.e., CBR has an average rate 80% higher than VBR encoding for each of the traces. Since each of our VBR traces has an average bit rate of 2 Mbps, each of the CBR videos has a bit rate of 3.6 Mbps. With a known CBR rate, channel rate, it is easy to determine the start-up latency for CBR for the case $q = K$ and a geometric broadcast series [6].

The CBR start-up latencies are given in Table 2 for three link capacities. In Table 2 we also present the start-up latencies for buffered multiplexing of VBR-encoded video. (We use buffered multiplexing instead of JSQ prefetching because it requires less time for simulation; JSQ prefetching can give even better performance.) For the buffered multiplexing, we chose the *K* value and buffer size combination which gives the lowest delay while having a loss probability less than 10E-7 (essentially a negligible loss probability). We see that for each of the link capacities, our VBR multiplexing scheme has reduced the start-up latency by more than a factor of 4.

C (Mbps)	*latency CBR*	*latency VBR*
85	35.6	7.3
145	7.1	1.7
205	3.4	0.2

Table 2: Latency in minutes of CBR and VBR video.

The dramatic reduction in start-up latency is primarily due to the fact that the latency decreases exponentially fast with *K*, the number of segments in the broadcast series. The lower average rate of VBR allows us to increase *K*, and thereby obtain significant reductions in start-up latency.

References

[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. A permutation-based pyramid broadcasting scheme for video-on-demand systems. In *Proc. of the IEEE Int'l Conf. on Multimedia Systems* '96, Hiroshima, Japan, June 1996.

[2] K. Almeroth and M. H. Ammar. The use of multicast delivery to provide a scalable and intercative video-on-demand service. *IEEE Journal on Selected Areas in Communications*, 16(6):1110-1122, 1996.

[3] 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, Departments of Electrical Engineering and Computer Science, Stanford University, August 1996.

[4] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proc. of ACM Multimedia*, pp. 15-23, San Francisco, California, October 1994.

[5] M. W. Garret and A. Fernandez. Variable bit rate video bandwidth trace using MPEG code. Available via anonymous FTP from thumper.bellcore.com., Nov 4, 1994.

[6] K. A. Hua, Y. Cai, and S. Sheu, A Client-Centric Approach to designing periodic broadcast schemes. Technical Report CS-TR-98-02, School of Computer Science, University of Central Florida, Orlando, Florida, January 1998.

[7] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on demand systems. In *Proc. of the ACM SIGCOMM'97*, Cannes, France, September 1997.

[8] M. Krunz, R. Sass, and H. Hughes. Statistical characteristics and multiplexing of MPEG streams. In *Proceedings of the IEEE INFOCOM '95 Conference*, pp.455-462, April 1995.

[9] T. V. Lakshman, A. Ortega, A. R. Reibman.VBR Video: Trade-offs and potentials. *Proceedings of the IEEE*, vol. 86, no. 5, May 1998, pp. 952-973

[10] W. Luo and M. El Zarki. Analysis of error concealment schemes for MPEG-2 video transmission over ATM based networks. In *Proceedings of SPIE Visual Communications and Image Processing 1995*, Taiwan, May 1995.

[11] 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, Augast 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.

[13] M. Reisslein, K. W. Ross, and V. Verillotte. A decentralized prefetching protocol for VBR video on demand. In D. Hutchison and R. Schafer, editors, Multimedia Applications, Services and Techniques-ECMAST'98(Lecture Notes in Computer Science Vol. 1425), pages 388-401, Berlin, Germany, May 1998. Springer Verlag. Extended version available at http://www.eurecom.fr/~ross/.

[14] O.Rose. Statistical properties of MPEG video traffic and their impact on traffic modelling in ATM systems. Technical Report 101, Univercity of Wuerzburg, Institute 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] W. D. Sincoski, System architecture for a large scale video on demand service. *Computer Networks and ISDN systems*, vol. 22 1991.

[16] W. S. Tan, N. Duong, and J. Princen. A comparison study of variable bit rate versus fixed bit rate video transmission. In Australian Broadband Switching and Services Symposium, pp. 134-141, 1991.

[17] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *Multimedia Systems*, 4(4):197-208, Augast 1996.

Appendix

JSQ Prefetching Implementation Issues for Periodic Broadcasting

Here we briefly outline an efficient implementation of the JSQ prefetch protocol. We initially exclude the prefetch delay refinement. We maintain the parameters of the *MK* ongoing video streams in an array consisting of *MK* records. Each record stores the parameters pertaining to one virtual buffer; in particular we need to keep track of $p_t(m, k)$. The record also contains a pointer to another record. This allows us to arrange the records in a singly linked list. The list is maintained such that the index $p(m, k)$ is ascending as one moves down the list, that is, the virtual buffer with the smallest number of prefetched frames is on the top. In each frame period we start by considering the virtual buffer on the top of the list. We first check whether condition (13) is satisfied. If this condition is satisfied we transmit the frame and increment *p(m, k)*. Next, we check whether the successor in the list has a smaller *p(m, k).* If not, we try to prefetch the next frame. This is repeated until *p(m, k)* is larger than that of the next virtual buffer in the list. At that point we rearrange the pointer references to maintain the order of the list. After the order has been restored we start over, trying to prefetch for the virtual buffer on top of the list. If we find at any point that (13) is violated, we skip the virtual buffer and try to prefetch for the successor in the list, that is, we prefetch no longer for the virtual buffer on top of the list, but move down the list as we skip over virtual buffers. At the end of the slot we decrement by one all *p(m, k)*'s that are larger than or equal to one. Note that this does not affect the ordering of the list.

We now focus on the implementation issues that arise when adding the prefetch delay refinement to the JSQ protocol. First of all, we see from Figure 7 that for the last d_{pre} frame periods of any segment *(m, k)* we already prefetch frames for the next broadcast of this very segment into the virtual buffer while we are still removing frames for the current broadcast of the segment form the virtual buffer. Furthermore, it is possible that during the same period we are still prefetching frames for the current broadcast of the segment into the virtual buffer. This occurs if the segment has not been prefetched in its entirety into the virtual buffer by the time prefetching for the next broadcast starts. The challenge is to manage the virtual buffers correctly. We have implemented the following simple solution. For each stream *(m, k)* we maintain two virtual buffers. The virtual buffers are in either one of three states, "ignore", "prefetch only" or "prefetch and consume". The server ignores a virtual buffer in the “ignore” state; frames are neither prefetched nor consumed. As we shall see shortly a virtual buffer in the "ignore" state is always empty. The server prefetches frames for a virtual buffer in the "prefetch only" state but frames are not consumed, that is, we do not decrement the *p(m, k)* of a virtual buffer in the "prefetch only" state at the end of the slot. The server prefetches frames for a virtual buffer in the "prefetch and consume" state and also decrements the *p(m, k)* at the end of the slot if the virtual buffer holds one or more frames. The virtual buffers change states according to a specific timing policy; this timing policy is illustrated in Figure 10. The first virtual buffer of a stream *(m, k)* is in the "prefetch only" state from time zero up to time d_{pre} (in frame periods). The virtual buffer is then in the "prefetch and consume" state for the subsequent $N_k(m)$ frame periods. After that, the virtual buffer is in the "ignore" state for $N_k(m) - d_{\text{pre}}$ frame periods. Then, the "prefetch only" - "prefetch and consume" - "ignore” cycle starts over. The second virtual buffer of the stream *(m, k)* is initially in the "ignore" state for $N_k(m)$ frame periods and starts then its regular "prefetch only" - "prefetch and consume" – “ignore" cycle. One caveat of this approach is that it works only when the prefetch delay is shorter than the first segment, that is, when $d_{\text{pre}} \leq N_1$. If this condition is not satisfied the length of the "ignore" period becomes negative. In essentially all cases of practical interest, however, the condition is satisfied.

Another issue that arises from the addition of the prefetch delay refinement to the JSQ protocol is that decrementing only the *p(m, k)*'s of the virtual buffers in the "prefetch and consume" state destroys the order of the list. To see this, consider any two virtual buffers *a* and *b* with $p(a) = p(b) \geq 1$ and suppose that *a* is succeeded by *b* in the linked list. Furthermore, suppose virtual buffer *a* is in the "prefetch only" state while virtual buffer *b* is in the "prefetch and consume" state. Also, suppose that we have reached the end of the slot and the decrementing step in the JSQ algorithm is executed next. Since virtual buffer *a* is in the "prefetch only" state its *p(a)* remains unchanged. Virtual buffer *b*'s *p(b)*, however, is decremented by one, destroying the order of the linked list. The order must be restored before the JSQ algorithm starts

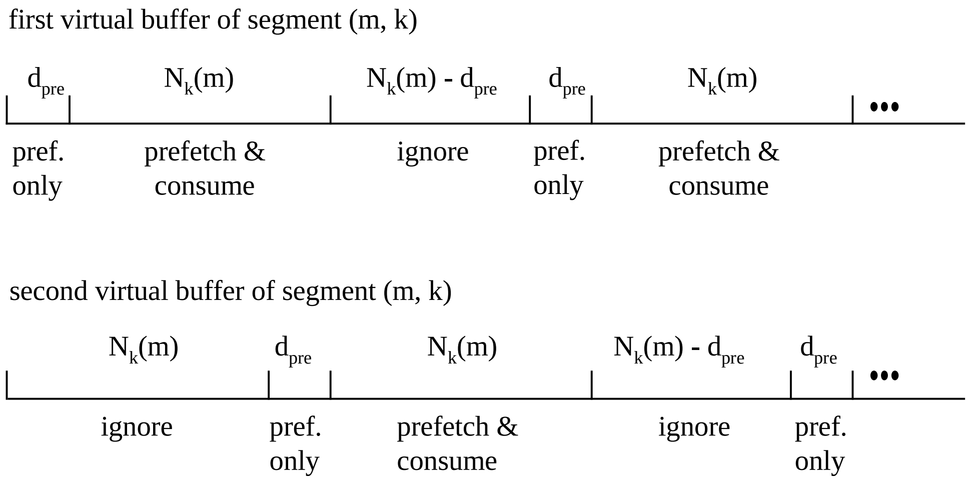

Figure 10: Timing policy of state changes of the two virtual buffers of segment (m, K).

over in the next slot. One approach to restore the order is to maintain a doubly linked list and rearrange the pointer references whenever a virtual buffer ends up with a smaller *p(m, k)* than its predecessor after decrementing. Another approach is to insert a sorting routine after the decrementing step in the JSQ algorithm and relink the list after the records have been sorted in increasing order of *p(m, k)*. Note that we don't have to move the records around in the array but can sort indices into the array instead. We have implemented the sorting approach. We found in our numerical experiments that it takes roughly 2 msec to execute the JSQ algorithm in each frame period for 70 video streams on a SUN Ultra 170 workstation.