Multimedia Systems 9: 503–516 (2004)
Digital Object Identifier (DOI) 10.1007/s00530-003-0123-2

Multimedia Systems
© Springer-Verlag 2004

Periodic broadcasting with VBR-encoded video*

Martin Reisslein[1], **Despina Saparilla**[2], **Keith W. Ross**[3]

[1] Department of Electrical Engineering, Arizona State University, P.O. Box 875706, Tempe, AZ 85287–5706, USA (e-mail: reisslein@asu.edu)
[2] Cisco Systems, Bedfont Lakes, Feltham TW14 8HA, UK (e-mail: saparill@cisco.com)
[3] Polytechnic University, Department of Computer and Information Science, Brooklyn, NY 11201, USA (e-mail: ross@poly.edu)

Abstract. We consider designing near video on demand (NVOD) systems that minimize startup latency while maintaining high image quality. Recently nonuniform segmentation has been used to develop periodic broadcasting techniques for NVOD. These techniques give significant reductions in startup latency as compared with more conventional uniform segmentation. Essentially all of these schemes assume, however, that the videos are CBR-encoded. Since a CBR-encoded video has a larger average rate than an open-loop VBR encoding 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 that are based on smoothing, server buffering, and client prefetching. Two key but conflicting performance measures exist when using VBR video: latency and packet loss. By introducing small additional delays in our multiplexing schemes, our trace-based numerical work shows that the schemes can achieve nearly 100% link utilization with negligible packet loss. When the ratio of the CBR rate to the VBR average rate is a modest 1.8, startup latency can be reduced by a factor of four or more for common scenarios.

Keywords: Near video on demand – Prefetching – Statistical multiplexing – Variable bit rate video

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 stored on central video servers. Requested videos are transmitted to a large population of clients through a network (e.g., cable, ADSL, or LAN), and a distinct stream is dedicated to each user. True VoD is referred to as user centered since server and network bandwidth is strictly divided among the system's users [1,30]. 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.

To provide scalable VoD, various techniques based on a data-centered approach have been developed in which the server divides its bandwidth among distinct video objects 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 [3,28]. Techniques in which many clients share a common server stream provide *near* VoD (NVOD). With NVOD, users experience a delay of the order of seconds to tens of minutes before the commencement of the video of their choice. In some NVOD techniques, this startup 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 [2,6,7,27]. In another set of NVOD 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 [1,9,13,14,30].

When the number of users is large, periodic broadcasting can be an efficient means to distribute stored video. Periodic broadcasting scales nicely, as the startup latency is completely independent of the number of clients. The startup latency depends, however, on the particular periodic broadcasting scheme and the number of videos that are broadcast. In broad terms, the fewer the number of videos, the greater the number of copies of each video that can be broadcast and the lower the initial startup latency. Fortunately, for movies on demand, a large fraction of the demand is typically for the 10 to 20 most popular movies [7].

One simple periodic broadcasting scheme is to broadcast multiple entire copies of each video, with a new copy broadcast every fixed interval of time (e.g., a new copy of *Star Wars* broadcast every 20 min) [6]. We refer to such schemes as periodic broadcasting with uniform segmentation. In such a scheme, the maximum startup latency experienced by a user is equal to the length of the video divided by the number of copies broadcast. This latency can be long for full-length MPEG2-

* A shorter version of this paper appeared in *Proceedings of IEEE Infocom '99*, New York, March 1999, pp 464–471.
Correspondence to: M. Reisslein (http://www.eas.asu.edu/∼mre)

encoded movies sent over channels on the order of 100 Mbps. For example, when ten movies, each encoded at 3 Mbps and each 2 h long, are broadcast over a 100-Mbps channel, the maximum startup latency is 40 min. Beginning with a seminal paper [30], a number of nonuniform segmentation schemes have recently been proposed [1,9,13,14,30]. Loosely speaking, these schemes reduce the initial startup latency by broadcasting the earlier portions of the video more frequently and the latter portions less frequently.

Essentially all of the existing work on NVODsystems with periodic broadcasting is based on the assumption that the videos are constant bit rate (CBR)-encoded [1,9,13–15,21, 30]. (We note that the tailor-made transmission scheme [4], which was developed independently of this work, also accommodates VBR video. VBR video is also accommodated by the periodic broadcast schemes with adaptive segmentation [18] that followed the conference presentation [26] of a preliminary version of the work presented in this article.) 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 the 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 [5,17].) 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 to its burstiness. Nevertheless, for a given movie or sporting event and *for the same quality level*, the average bit rate of a CBR MPEG encoding is typically two times or more the average bit rate of a VBR MPEG encoding [5,29]. Therefore, with VBR video there is potential for increased system efficiency. We note that to our knowledge our study is the first to investigate NVOD systems using periodic broadcasting of *VBR-encoded video*.

Although nonuniform segmentation can greatly reduce startup latency, for many practical circumstances the startup latency remains unacceptably high for CBR-encoded video. When a NVOD system broadcasts full-length MPEG2-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 2 h long, are broadcast over a 100-Mbps channel, the maximum initial startup latency is more than 17 min. In this paper we develop nonuniform segmentation schemes with VBR-encoded video that significantly reduce the initial startup latency without appreciably degrading image quality. In particular, for situations of practical interest, as the one described above, the startup 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 startup 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 to develop a NVOD scheme that uses VBR-encoded video and yet has low packet loss, on the order of 10^{-6} or less. (Losses this small can be effectively hidden with simple techniques for error concealment by postprocessing at the decoder; see Sect. V of [31].) We investigate a number of different schemes in this paper and find that join-the-shortest-queue (JSQ) prefetching gives the best performance. The JSQ prefetching policy is akin to the Earliest Deadline First (EDF) scheduling policy in that the sender selects the frame with the earliest playback deadline for transmission. Our numerical work demonstrates that periodic broadcasting of VBR-encoded video with JSQ prefetching achieves small startup latencies and extremely small loss probabilities.

This paper is organized as follows. In Sect. 2 we show how nonuniform segmentation can be combined with bufferless statistical multiplexing to create a NVOD scheme using VBR-encoded video. We present a methodology that explores the trade-off between startup latency and packet loss. In Sect. 3 we study a specific segmentation scheme and present results from trace-driven simulations for bufferless multiplexing. The bufferless multiplexing scheme does not provide sufficiently low loss probabilities, but it does set the stage for a series of more sophisticated schemes described in Sect. 4 that offer increasingly 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 JSQ prefetching and leads to yet further improvements. We provide extensive numerical examples that show that the aforementioned schemes can lead to dramatic reductions in initial startup latency while keeping the loss probability negligible. In Sect. 5 we show that our VBR multiplexing schemes can dramatically reduce the CBR startup latency.

1.1 Overview of nonuniform segmentation techniques

The first nonuniform segmentation technique referred to as pyramid broadcasting [30] 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 broadcast sequentially on the i-th logical channel. Since the first segments of the videos are the smallest in size, they are also broadcast the most frequently. Consequently, the startup latency for any video is significantly reduced. The reduced startup latency achieved in [30] 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 subchannel 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 startup latency, the skyscraper broadcasting technique [14] 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. In [13] Hua et al. propose yet another segmentation scheme that results in further reduction in startup latency relative to [14] without compromising storage and disk bandwidth costs. This 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. Finally, Gao et al. [9] develop a rigorous analytical framework for studying periodic broadcasting schemes with CBR video. Based on their analytical framework Gao et al. develop the Greedy Disk-conserving Broadcasting (GDB) scheme, which achieves small startup latencies even for clients with limited pipelining capabilities.

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 broadcast and $N(m)$ the number of frames in the m-th video. All videos are VBR-encoded. 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 $x_n(m),\ n = 1,\ldots,N(m),\ m = 1,\ldots,M$ denote 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 startup 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 [9,13,14]. The choice of download strategy depends on the ability of the client to employ pipelining, i.e., on the 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 [1,9, 13,14,30]. 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 units (whereby one "unit" corresponds to a certain integer number of video frames), the second segment of e_2 units, etc. Without any loss of generality, set $e_1 = 1$ unit. 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 $N_i(m) = e_i N_1(m),\ i = 2,\ldots,K$.

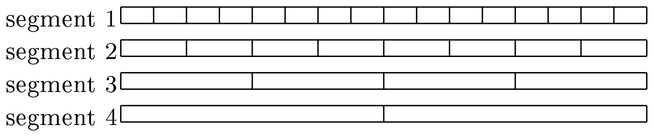

Fig. 1. Broadcasting strategy for geometric series with $e_k = 2^{k-1}$

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

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

Thus, a broadcast series $[e_1, e_2, \ldots, e_{K-1}, e_K]$ and video length $N(m)$ completely specify 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. (Throughout this paper we focus on a playback-only system, i.e., we do not consider client interactions, such as pause and temporal jumps.) This requirement is referred to as the continuity condition. Note that the broadcast series specifies the transmission schedule in terms of numbers of video frames, independent of whether the frames have constant or variable sizes. Hence there is no difference between the continuity condition for CBR-encoded and VBR-encoded video. [We do note that the specification of the broadcast series in terms of numbers of video frames assumes that all video frames of a given video have the same frame period (display time), as is typical for VBR-encoded video.] Whether a certain scheme satisfies the continuity condition, given F and $N(m)$, depends on the value of q and the specific form of broadcast series used. For example, consider $q = 1$, i.e., the client can download frames from only one stream at a time. In this case, the uniform broadcast series $[1, 1, \ldots, 1]$ is the only type of series that results in a feasible delivery scheme. When $q = 2$, the series of increasing terms given in [9,13,14] satisfy the continuity requirement. Finally, in the case where $q = K$, a larger set of broadcast series results in feasible delivery schemes. For example, the geometric series $[1, 2, \ldots, 2^{K-1}]$ meets the continuity condition.

As a specific example, consider the continuity condition for $[1, 2, \ldots, 2^{K-1}]$. The broadcasting strategy for this series is illustrated in Fig. 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 $N_{i+1} = 2 \cdot N_i(m)$, the broadcasts of the segments either begin or end at the same time. In the case where 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 where the broadcasts end at the same time, a broadcast of segment $i + 1$ immediately follows that of segment i, and continuity is maintained.

Startup 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 equals the broadcast duration of the segment. We let $L(m)$ indicate the startup latency for the m-th video. We have $L(m) = N_1(m)/F$. For a general broadcast series $[e_1, e_2, \ldots, e_{K-1}, e_K]$, where $e_1 = 1$, the startup latency is given by

$$L(m) = \frac{N(m)}{F \cdot \sum_{i=1}^{K} e_i}. \qquad (2)$$

Note that the startup latency of a periodic broadcasting scheme is decreasing in K. Also, note that for a given value of K the startup latency is decreased when a fast growing broadcast series is utilized.

With VBR video there are two performance measures, startup latency and loss probability. As we shall see, there is a trade-off between these two measures. We consider the commonly used *information* loss probability, which is defined as the long-run fraction of encoding information (bits) lost during network transport (the periodic broadcasting in our context). An alternative commonly used metric is the frame loss probability, which is defined as the long-run fraction of (the number of) video frames that suffer losses (i.e., a part of the frame or the entire frame is lost) during network transport. We use the information loss probability because error concealment techniques (see Sect. V of [31] and references therein) enable clients to decode and display partially received video frames, albeit with a reduced quality. 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 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:

$$y_t(m, k) = x_m(j),$$

where j is given by

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

Note that j represents the index for the frame of the m-th video (i.e., $j = 1, \ldots, N(m)$) 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 $m = 1, \ldots, M,\ k = 1, \ldots, K$, y_t is computed as $y_t = \sum_{m=1}^{M} \sum_{k=1}^{K} y_t(m, k)$. In our initial 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, i.e., if $y_t > \frac{C}{F}$. Thus the long-run fraction of bits lost is

$$\begin{aligned} P_{\text{loss}} &= \lim_{T\to\infty} \frac{\text{\# of bits lost up to frame time } T}{\text{total \# of bits sent up to frame time } T} \\ &= \lim_{T\to\infty} \frac{\sum_{t=1}^{T} (y_t - C/F)^+}{\sum_{t=1}^{T} y_t}, \end{aligned} \qquad (3)$$

where $(x)^+ := \max(0, x)$.

Small startup latency and small loss probability are conflicting objectives. The startup latency is decreasing in 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 loss probability also increases with K. In the next section, we present numerical results from the simulation of a specific periodic broadcasting scheme that illustrate the trade-off between startup latency and loss probability.

3 Periodic broadcasting with a geometric 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). To illustrate the use of VBR video with periodic broadcasting, we focus our numerical work on one download strategy and broadcast series. Specifically, we use $q = K$, i.e., the client can download each of the K segments at their next occurrence, and the geometric series $[1, 2, \ldots, 2^{K-1}]$ to segment the videos. It is shown in [9] that the geometric broadcast series is the fastest growing broadcast series that satisfies the continuity condition; the geometric broadcast series therefore gives the smallest startup latencies (Eq.2) and the fewest segments. (We note that other, slower growing broadcast series result for the same small startup latency in more segments for each video and thus a correspondingly larger total number of streams (which would require a correspondingly larger link capacity). Since a larger number of streams generally improves the statistical multiplexing effect, this scaling would tend to result in better performance for our VBR schemes than is reported here for the geometric series.) Recall from Sect. 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, approaches presented in [9, 14], in which segment sizes are restricted to a maximum value, can instead be employed.

We obtained 7 MPEG1-encoded movies from the public domain [10, 16, 24]. The trace of each movie gives the number of bits in each frame. The seven encoded movies were used to create ten "pseudotraces" each 160,000 frames long. Table 1 summarizes the statistics of the resulting traces. The ten traces used in the numerical study were created from the seven movies in the following manner. The first five traces were created using the encoded movies in [24]. Since each of the original movies is 40,000 frames long, 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 was created by repeating four times the first 40,000 frames of the MPEG encoding obtained from [16] and then manipulated such that its average bit rate is 2 Mbps. Finally, the MPEG encoding obtained from [10] was first divided into four parts of 40,000 frames each. The resulting movie segments were repeated four times to create four different trace sequences of 160,000 frames. The trace sequences were finally multiplied by constants to create the last four traces of Table 1 with aver-

Table 1. Trace statistics

	Frames		GoPs	
Trace	Peak/mean	St. dev (Mbits)	Peak/mean	St. dev (Mbits)
Bond	10.1	2.11	4.2	1.10
Lambs	18.4	3.06	3.3	1.57
Mr. Bean	13	2.34	5.0	0.48
Soccer	6.9	1.91	3.7	1.87
Terminator	7.3	1.86	2.8	2.13
Wiz. of oz	8.4	2.48	3.2	3.48
Star wars 1	10.9	2.45	3.7	2.37
Star wars 2	13.2	2.34	4.4	2.57
Star wars 3	12	2.31	3.1	2.77
Star wars 4	8.5	2.14	3.2	2.96

age bit rates equal to 2 Mbps. Although the ten pseudotraces are not traces of actual movies, we believe that they reflect the characteristics of MPEG2-encoded movies (highly bursty, long-range scene dependence, average rate about 2 Mbps).

We expect that future NVOD systems will offer videos with higher average bit rates (e.g., HDTV quality video) and have proportionally higher link rates C. With this scaling the performance characteristics of the NVOD system are approximately equal to the performance characteristics obtained for the 2-Mbps average bit rate videos and link rates C considered in this study. The periodic broadcast schemes studied in this paper do not rely on the assumption of equal average bit rates and accommodate heterogeneous average bit rates without modifications. When supporting videos with heterogeneous average bit rates, both the basic statistical multiplexing [23] and JSQ prefetching mechanisms [20] studied here in the context of periodic broadcasting distribute the losses approximately proportionally to the average bit rates among the streams. That is, streams with a higher average bit rate experience proportionally larger losses, and vice versa for lower average bit rate streams. Both the studied multiplexing and prefetching mechanisms may be combined with fairness mechanisms that distribute the losses differently, e.g., prioritize the drop of video information according to its contribution to the perceived video quality, which is currently a topic of ongoing research. In summary, we use $M = 10$ VBR-encoded videos ($F = 25$ frames/s) each of which has 160,000 frames, i.e., $N(m) = N = 160,000$ frames, and a length of approximately 107 min. As illustrated in Table 1, the traces used in the numerical study have high peak/mean ratios and standard deviations.

Our numerical study of the periodic broadcasting scheme focuses on startup latency and loss probability. The startup latency for the geometric broadcast series with $e_k = 2^{k-1}$ is given by (see Eq. 2)

$$L = \frac{N}{F \cdot (2^K - 1)} \tag{4}$$

for each video. The loss probability is given by the expected fraction of bits lost from the video streams during broadcasting (Eq. 3). In general, to obtain an accurate approximation of P_{loss}, it is necessary to simulate the periodic broadcasting scheme over a large number of frame times. In our case, however, the use of the geometric series $[1, 2, \ldots, 2^{K-1}]$ 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. Thus we compute the loss probability P_{loss} as follows:

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

Note that this equation applies to all broadcasting series where the length of the longest segment is an integer multiple of the lengths of the shorter segments. For broadcast series that do not satisfy this property, P_{loss} can be estimated in similar fashion by replacing the length of the longest segment by the smallest common multiple of the lengths of the segments.

To obtain confidence intervals of the loss probability, we perform multiple independent replications of the broadcast in which the broadcast traffic pattern varies. To allow for replications using the available set of traces, we introduce a random shift in what we consider the starting point of each trace. In each replication of the broadcasting scheme, ten random numbers are drawn from a uniform distribution in the range $[1,\ 160000]$ to determine random starting points for each of the traces. Starting from the 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. We run the simulations until the 90% confidence intervals of the loss probability are less than 10% of the point estimate.

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 onto a bufferless link. (In practical systems a multiplexer buffer that can hold MK packets is required for packet-scale queueing. However, the maximum packet-scale queueing delay $M \cdot K \cdot \text{max\_packet\_size}/C$ is negligible. To see this, note that for stability $C/(MK)$ must be larger than the average of the average rates of the video segments, which is on the order of 2 Mbps or larger for MPEG2-encoded video.) The results illustrate the startup latencies and loss probabilities achieved by different values of K. We only consider values of K that generate startup latencies in the range from 0–16 min and loss probabilities below 0.1. The startup 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, the startup latency decreases exponentially with K. It is easily seen from Eq. 4 that for startup latencies below 2 min, K must be at least 6. Startup latencies below half a minute result from values of K that are at least 9. As the number of segments K increases, the loss probability increases.

In Fig. 2, the loss probability, P_{loss}, is plotted as a function of the startup latency for three different link capacities. (The link capacities are chosen to examine the statistical multiplexing for a range of system utilizations. Note that for the considered traces with an average bit rate of 2 Mbps, the aggregate

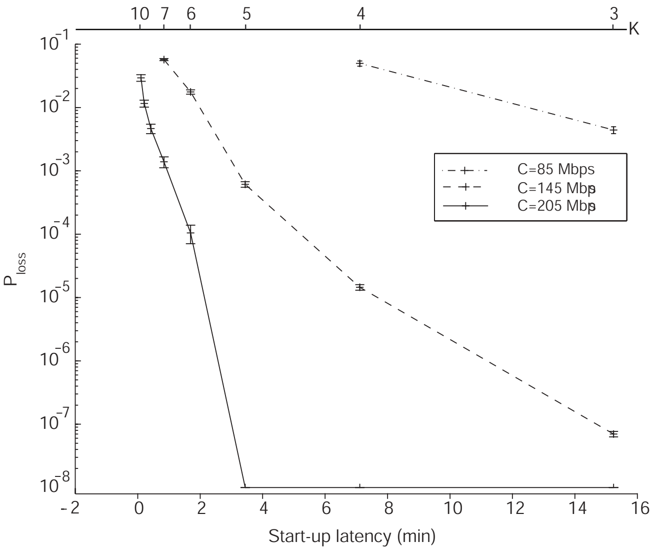

Fig. 2. Bufferless statistical multiplexing

obtained by simultaneously broadcasting K segments for each of the ten videos has (on the average over many experiment replications) an average of $20 \cdot K$ Mbps, which corresponds to a long-run average system utilization of $20 \cdot K/C$.)

The figure shows the 90% confidence intervals for P_{loss}. Each point on the curve for a fixed link bandwidth corresponds to a specific value of K. The curve for $C = 85$ Mbps is spanned by the points for $K = 3$ and $K = 4$ from right to left (corresponding to average system utilizations of 70% and 90%, respectively), while the curve for $C = 145$ Mbps gives points for $K = 3, 4, 5, 6$, and 7. Finally, the curve for $C = 205$ Mbps gives points for $K = 6, 7, 8, 9$, and 10. Depending on the available link capacity, startup latencies below 16 min and loss probabilities below 0.1 are achieved for K values between 3 and 10. Smaller K values result in startup latencies that exceed 16 min, while higher values generate loss probabilities above 0.1. Figure 2 illustrates the trade-off between P_{loss} and startup latencies with varying K. For startup latencies below 2 min, the loss probability is high (i.e., on the order of 10^{-4} or higher) for all studied link bandwidths. While this loss-latency trade-off 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 min, 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 to improve performance.

4 Improving performance

In this section we examine three different methods for reducing loss. The first is GoP smoothing of the VBR traces prior to broadcasting. The second is buffered statistical multiplexing, that is, we add a finite buffer at the server link. The third uses JSQ prefetching of video frames during periods of time when the shared link's bandwidth is underutilized.

4.1 GoP smoothing

In this section we investigate the effect of Group of Pictures (GoP) smoothing on the performance of the broadcasting scheme. GoP smoothing is a very simple smoothing scheme and effectively smoothes the small time-scale variation that is due to the interframe dependence in MPEG encodings. GoP smoothing is done in the video server prior to sending the video traffic into the network. The GoP smoothed streams are then statistically multiplexed onto the shared bufferless link of capacity C. We first obtain results for the case when the video traces are smoothed over each GoP period for link capacities ranging from 85 to 205 Mbps. The results are plotted in Fig. 3.

The total startup latency shown in the plot is the sum of the maximum access time for the first video segment (given by Eq. 4) and the delay introduced due to smoothing over one GoP period. With a GoP size of 12 frames and a frame rate of 25 frames per second, the additional startup delay introduced due to smoothing equals 0.48 s.

We observe a significant reduction of the loss probability due to GoP smoothing for all three link capacities. Note that 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 s.

We now focus on the case when $C = 145$ Mbps. Our numerical study aims at examining the effect of further smoothing on the loss probability. Instead of smoothing the video traces over one GoP period, we employ smoothing over a larger number of GoP periods. The results are shown in Fig. 4.

We concentrate on values of K equal to 6 and 7 since we have not observed any losses in very long simulations for smaller K. (We conservatively plot a loss probability of 10^{-8}, with a slight vertical offset to avoid overlapping lines, when we

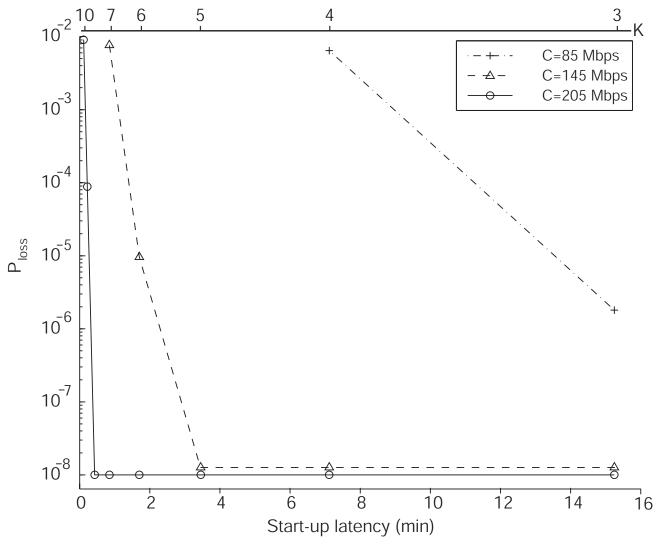

Fig. 3. Bufferless multiplexing with smoothing over one GoP period

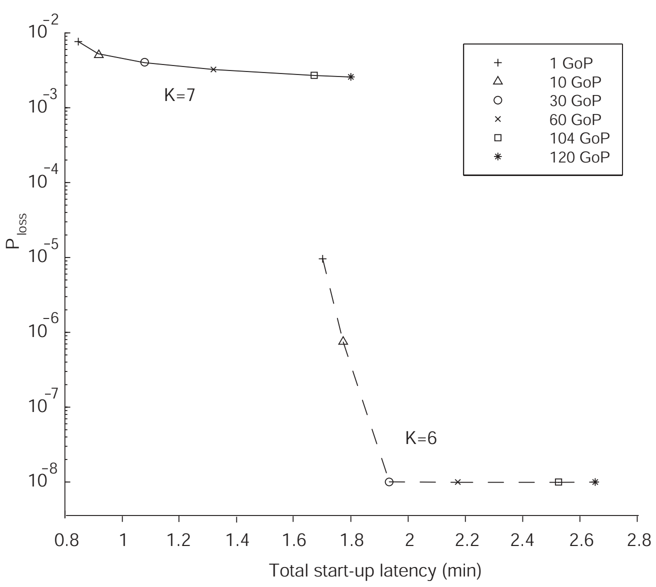

Fig. 4. Smoothing over many GoP periods ($C = 145$ Mbps)

have not observed any losses after running 10,000 independent replications, which corresponds to the simulated transmission of $9 \cdot 10^{15}$ bits.)

Let us first consider the cluster of points in the 1.6–2.8-min interval on the latency scale. These points correspond to different smoothing policies for $K = 6$. Clearly, smoothing over intervals of 10 or 30 GoP periods results in a considerable decrease in P_{loss}. Smoothing over intervals of 60 to 120 GoP periods introduces a longer smoothing delay without further reducing P_{loss}. Thus, 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 startup latencies with no further reduction of P_{loss} as *dominated*. Now consider the leftmost cluster of points in the 0.5–2-min latency interval for which $K = 7$. These points are nondominated in the sense that there is always a reduction of 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.

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, observe from Fig. 4 that division of the video files into seven segments when smoothing over 120 GoP periods is implemented, results in higher latency (and larger loss probability) than division into six segments with smoothing over 1 GoP period. Hence the scheme with $K = 6$ and 1 GoP smoothing dominates the scheme with $K = 7$ and 120 GoP smoothing. In conclusion, smoothing over a moderate number of GoP periods does not have an adverse effect when low startup latencies are desirable.

As elaborated in the appendix, we do not investigate more sophisticated noncollaborative smoothing schemes, such as optimal smoothing [25]. It was shown in [22] that prefetching based on the join-the-shortest-queue (JSQ) principle, which achieves collaborative smoothing, outperforms optimal smoothing. We focus, therefore, on the JSQ principle and study it in the context of periodic broadcasting in Sect. 4.3.

4.2 *Buffered statistical multiplexing*

In this section we consider buffered statistical multiplexing (with no smoothing) of the video streams by introducing a finite buffer of size B at the server link. In this scenario the server sends out the video traffic unsmoothed. If the aggregate traffic exceeds the link capacity C, it is buffered in front of the link, up to the capacity B of the buffer. Arriving traffic that finds the buffer full is dropped (lost). We vary B in the range of 72 to 8700 Mbits, corresponding to an additional startup delay of 0.5 to 60 s. (The added delay is equal to B/C seconds, which is the maximum possible delay that can be introduced due to buffering.)

The results are shown in Fig. 5 for $C = 145$ Mbps. The total startup 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. To facilitate the comparison of results, we include the case of bufferless statistical multiplexing. Figure 5 shows results for $K = 6$ and $K = 7$.

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, a startup latency of approximately 1.7 min ($K = 6$) 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 values higher than 72 Mbits, when $K = 6$, results in dominated points. When $K = 7$, increasing the size of the buffer achieves consistent reduction of P_{loss}. Note, however, that the reduction of P_{loss} becomes less significant as the buffer sizes increase. The most dramatic reduction occurs for an increase of the buffer size from 0 to 72 Mbits. Increasing the buffer size from 4350 to 8700 Mbits, however, results in a significant added delay of 0.5 min for a more moderate reduction of P_{loss}. Using a buffer of 8700 Mbits when $K = 7$ generates a dominated point (be-

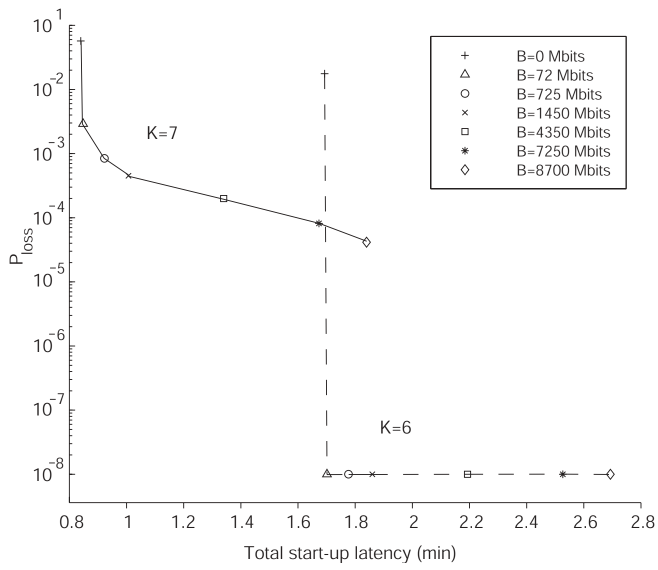

Fig. 5. Buffered statistical multiplexing ($C = 145$ Mbps)

cause both startup latency and P_{loss} are smaller for $K = 6$ and $B = 72$ Mbit). Thus, in this example, utilizing buffers that introduce (maximum) delays longer than 30 s is not desirable. To limit loss it is instead preferable to use a smaller K.

4.3 Join-the-shortest queue prefetching

The prefetching of video frames 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 link's bandwidth is underutilized. We develop a prefetching policy that selects the frames to be prefetched according to the join-the-shortest-queue (JSQ) principle. The JSQ prefetch policy is inspired by the Earliest Deadline First (EDF) scheduling policy. The EDF scheduling policy is known to be optimal among the class of nonpreemptive scheduling policies for a single link [11]. It is therefore natural to base the prefetch policy on the EDF scheduling policy. With JSQ prefetching the server selects the video frame with the earliest playback deadline for transmission. In other words, the server transmits the next frame to the playout queues with the smallest number of prefetched frames (i.e., the shortest queues).

We note that in [22] a JSQ prefetching protocol for the streaming of VBR-encoded video in a VoD system is described. The prefetching work in [22] focuses on aspects of a clientcentric JSQ prefetching protocol. In particular, a refined JSQ prefetching policy, which accommodates finite client buffers and client interactions, is developed. Also, admission control strategies, which incorporate criteria such as the long-run average link utilization and a large deviations estimate of the lost intracoded (I) frames, are devised. Finally, JSQ prefetching is compared with optimal smoothing [25]. The numerical work in [22] shows that JSQ prefetching gives better performance than the statistical multiplexing of video streams that have been smoothed using the optimal smoothing technique.

In this paper we study the JSQ principle in a very different context. We develop JSQ prefetching policies for data-centered near video on demand (NVOD) systems. We shall first develop and evaluate a baseline JSQ prefetching policy for data-centered video streaming. We shall then develop an important refinement for data-centered JSQ prefetching. This refinement gives the playout buffers some startup delay to build up a prefetched reserve before playback commences and at the same time satisfies the continuity condition for uninterrupted playback.

For the development of the baseline JSQ prefetch policy we introduce the concept of *virtual buffers*. We assign to each stream (m, k) a virtual buffer that tracks the buffer contents of a client that is tuned in to stream (m, k) at all times and repeatedly displays segment (m, k). We first describe in detail the JSQ policy's operation in the system of virtual buffers. We then explain how the system of virtual buffers relates to the actual NVOD system.

In explaining the details of the JSQ policy 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,

$$p_{t+1}(m, k) = [p_t(m, k) + \Delta_t(m, k) - 1]^+. \tag{6}$$

If the frame scheduled to be displayed at the end of the slot does not arrive in time, the virtual buffer is starved 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 Eq. 6.

During each slot of length $1/F$ seconds the server decides which frames to transmit from the MK ongoing streams. This is done according to the JSQ principle. 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. 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 each slot the server determines the stream (m^*, k^*) with the smallest $p(m, k)$ and checks whether

$$z + x_{\sigma(m^*,k^*)}(m^*) \leq C/F, \tag{7}$$

where $\sigma(m^*, k^*)$ is the frame (number) of video m^* considered for transmission. ($x_{\sigma(m^*,k^*)}(m^*)$ is the size (in bits) of the frame considered for transmission.) If Eq. 7 holds, we transmit the frame, increment $p(m^*, k^*)$ by one, and update z (by setting $z \leftarrow z + x_{\sigma(m^*,k^*)}(m^*)$). If Eq. 7 is violated, we remove the stream (m^*, k^*) from consideration and find a new stream (m^*, k^*) that minimizes $p(m, k)$. If Eq. 7 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 Eq. 7, we skip the corresponding stream and find a new stream (m^*, k^*). Once all streams have been skipped, we set $p(m, k) = [p(m, k) - 1]^+$ for all $m = 1, \ldots, M$ and $k = 1, \ldots, K$ and move on to the next slot. At the beginning of the new slot we reset z to zero and start over transmitting frames for the streams with the smallest $p(m, k)$.

In the real NVOD system, the server schedules the broadcast of the frames of the MK video streams as if they were being sent to the MK distinct virtual buffers. The clients tune in to a video, which is identified using some simple signaling protocol (which we do not discuss in any detail in this paper). Segments that arrive early are temporarily stored in the client, typically by writing them to disk. The video frames are then retrieved and displayed as their deadlines arrive. Note that prefetching does not interfere with the continuity condition.

To accommodate clients with a limited reception rate, we add a refinement to the JSQ prefetch policy. Suppose that the clients' reception rate is limited to R (bit/s). In this case, within each slot the server keeps track of $z(m, k)$, the total number of bits sent for segment (m, k) in the slot. Suppose the server is considering transmitting the frame $\sigma(m^*, k^*)$ of video m^*. It transmits the frame in the current slot if and only if Eq. 7 and

$$\sum_{k=1}^{K} z(m^*, k) + x_{\sigma(m^*,k^*)}(m^*) \leq R/F$$

are satisfied. This condition ensures that the server does not violate the reception constraint for any client. To simplify the discussion, for the remainder of this paper we shall assume that $R \geq C$.

How do the virtual buffers relate to the buffers in the clients of the NVOD system? When the geometric broadcast series $e_k = 2^{k-1}$ is used, the lengths of any two successive segments (m, k) and $(m, k+1)$ satisfy $N_{k+1}(m) = 2 \cdot N_k(m)$. Hence, each two segments either begin or end at the same time. First, consider the case where the two segments end at the same time, i.e., the ends of segments (m, k) and $(m, k+1)$ coincide. In this case the client starts to receive and display the next broadcast of segment $(m, k+1)$ immediately after segment (m, k) has ended. During the broadcast of segment $(m, k+1)$, the buffer contents of the client in the NVOD system and the buffer contents of the virtual buffer in the virtual buffer system are exactly the same. Hence, whenever loss occurs in the virtual buffer system, the NVOD system suffers exactly the same loss. In the case where the segments begin at the same time, 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 displayed. In other words, the display of segment $(m, k+1)$ is delayed by $N_k(m)$ frame periods by the client. This implies that whenever frame starvation occurs at the virtual buffer, loss is detected at the client $N_k(m)$ frame periods later. The long-run loss probability is therefore the same for both systems.

Table 2 gives the results of a simulation study of the JSQ policy with $C = 145$ Mbps.

We compare the JSQ results with the results obtained for bufferless statistical multiplexing in Sect. 3.1. Note that the JSQ policy runs over a bufferless link. We observe that the JSQ policy brings significant improvement over simply multiplexing the video streams onto the bufferless link. For $K = 7$ (i.e., a startup latency of 50.4 s) the loss probability drops from $5.7 \cdot 10^{-2}$ to approximately $3 \cdot 10^{-4}$ with JSQ prefetching.

We next develop a refinement of the JSQ policy that allows the virtual buffers and clients in the NVOD system to build up a reserve of frames over a certain period of time. We refer to the length of the period of time during which frames are prefetched but not consumed as the prefetch delay, denoted by d_{pre} frame periods. We refer to the refined JSQ policy as *JSQ prefetching with prefetch delay*. The total startup latency in this case is $L = (N_1 + d_{\text{pre}})/F$.

JSQ prefetching with prefetch delay of segments generated according to the geometric broadcast series $e_k = 2^{k-1}$ satisfies the continuity condition as illustrated in Fig. 6.

Consider any two successive segments (m, k) and $(m, k+1)$. Recall that the segments either begin or end at the same time. In the case when the segments begin at the same time, prefetching for the segments also starts at the same time

Table 2. Comparison of JSQ prefetching and bufferless statistical multiplexing

	P_{loss}	
Startup latency	JSQ prefetching	Bufferless mux
0.84 min (K = 7)	$3.0 \cdot 10^{-4}$	$5.7 \cdot 10^{-2}$
1.7 min (K = 6)	$6.0 \cdot 10^{-8}$	$1.8 \cdot 10^{-2}$

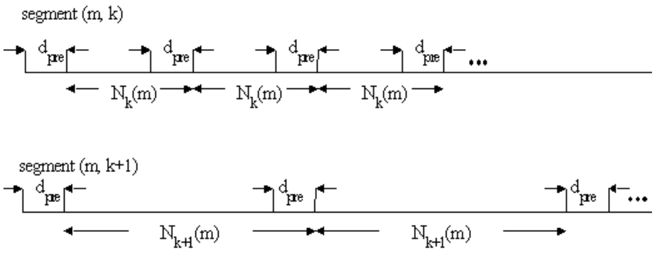

Fig. 6. JSQ prefetching with prefetch delay: prefetching starts d_{pre} frame periods before the first frame is consumed at the virtual buffer

(i.e., 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. When the segments (m, k) and $(m, k+1)$ end at the same time, 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 discussion of the implementation issues associated with the JSQ prefetching policy, we refer the reader to the appendix. In the appendix we outline efficient methods for precomputing the JSQ transmission schedule offline. Thus, only a table lookup is required during the NVOD system's operation.

We note that, even though we have discussed the datacentric JSQ prefetching without and with prefetch delay in the context of a geometric broadcast series, the developed JSQ prefetching schemes can be employed for any other broadcast series in analogous fashion. The only requirement is that for JSQ prefetching with prefetch delay the length of the prefetch delay d_{pre} must be shorter than the length of the shortest segment N_1, as detailed in the appendix.

Table 3 gives the results of a simulation of JSQ prefetching with prefetch delay for $C = 145$ Mbps and $K = 7$.

The introduction of the prefetch delay improves the loss probability significantly. For a prefetch delay of 10 s, the loss probability decreases from $3 \cdot 10^{-4}$ to $9.1 \cdot 10^{-5}$. Increasing the prefetch delay to 50 s results in a total startup latency of 100.4 s and a lower loss probability of $7.5 \cdot 10^{-6}$. Observe from Table 2, however, that JSQ prefetching without prefetch delay for $K = 6$ results in a total startup latency of 100.7 s and a loss probability equal to $6 \cdot 10^{-8}$. This indicates that to achieve a low loss probability, in the considered example, it is preferable to use a smaller K rather than a long prefetch delay. This observation parallels the conclusion of Sect. 4.2 on buffered multiplexing, which indicated that smaller K gives better performance than the use of very large buffers.

We now 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 the results from Figs. 3 and 5 and Table 3 (in conjunction with Table 2). The dominance curves specify the nondominated 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 Fig. 7, indicate that for similar latencies, JSQ prefetching gives the smallest loss probabilities.

Table 3. JSQ prefetching with prefetch delay

d_{pre} (s)	0	0.5	5	10	30	50
Total startup latency (min)	0.84	0.85	0.92	1.0	1.3	1.7
P_{loss}	$3.0 \cdot 10^{-4}$	$2.0 \cdot 10^{-4}$	$1.2 \cdot 10^{-5}$	$9.1 \cdot 10^{-5}$	$5.5 \cdot 10^{-5}$	$7.5 \cdot 10^{-6}$

We observe that the loss probabilities associated with buffered multiplexing can be an order of magnitude higher than the ones achieved 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 is clearly seen in Fig. 7, JSQ prefetching and buffered multiplexing is more effective in terms of limiting the loss probability than GoP smoothing.

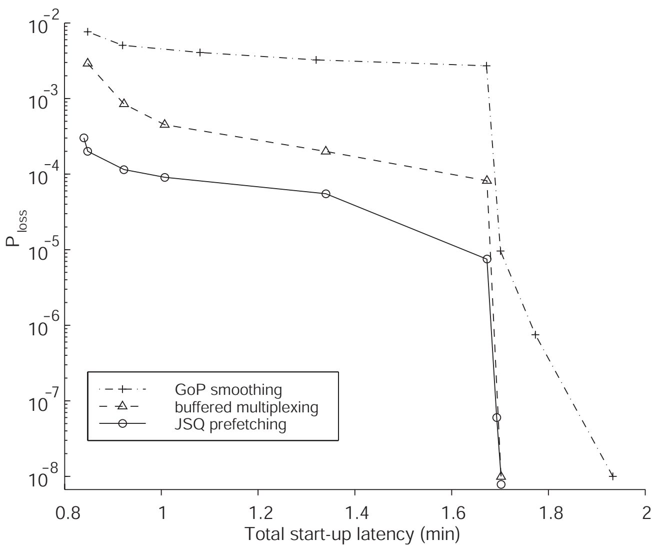

Fig. 7. Dominance curves for GoP smoothing, buffered multiplexing, and JSQ prefetching ($C = 145$ Mbps)

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-encoded and VBR-encoded video. 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 used 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 [5,29]. 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 and channel rate, it is easy to determine the startup latency for CBR for the case $q = K$ and a geometric broadcast series [13,9].

Table 4. Latency in minutes of CBR and VBR video

C (Mbps)	Latency of CBR	Latency of VBR
85	35.6	7.3
145	7.1	1.7
205	3.4	0.2

The CBR startup latencies are given in Table 4 for three link capacities. In Table 4 we also present the startup 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 that gives the lowest delay while having a loss probability less than 10^{-7} (essentially a negligible loss probability). We see that for each of the link capacities, our VBR multiplexing scheme has reduced the startup latency by more than a factor of 4.

The dramatic reduction in startup 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 startup latency.

Acknowledgements. This work has been supported partially by the National Science Foundation through grants NCR97-06495, Career ANI-0133252, and ANI-0136774 and in part by the State of Arizona through the IT301 initiative.

Eurecom's research is partially supported by its industrial partners: Ascom, Cegetel, France Telecom, Hitachi, IBM France, Motorola, Swisscom, Texas Instruments, and Thomson CSF.

We are grateful to Patrick Seeling of Arizona State University for assisting in the generation of the plots.

Appendix: JSQ prefetching implementation issues for periodic broadcasting

In this appendix we outline efficient methods for the offline computation of the JSQ transmission schedule for a NVOD system. Each week (or day) the NVOD provider could use these methods to test various potential movie profiles for the upcoming week. The provider could then determine which profiles it can support and which it cannot. It would then choose the profile from the set of feasible profiles that will most likely generate the most revenue.

This appendix is structured as follows. First, we describe an efficient method for precomputing the transmission schedule for JSQ prefetching without prefetch delay. Next, we discuss the computation of the transmission schedule for JSQ prefetching with prefetch delay. Finally, we briefly discuss the use of noncollaborative smoothing techniques, such as optimal smoothing [25], in NVOD systems.

JSQ prefetching without prefetch delay. Recall from Sect. 2 that $N_K(m),\ m = 1, \ldots, M$ denotes the length of the longest segment of video m in frame periods. It is natural to assume that the periodic broadcast of all M videos starts at the same time. It is then easy to see that the transmission schedule of the NVOD system is periodic. The length of the period is given by the smallest common multiple of the $N_K(m)$'s. Let P denote the length of the period in frame periods. For the special case that all $N_K(m)$'s are identical (i.e., $N_K(m) = N_K$), the length of the period of the transmission schedule is $P = N_K$. Because of its periodicity, the transmission schedule can be computed offline. In the following we outline an efficient method for precomputing the JSQ transmission schedule. For each of the transmission slots $t = 1, \ldots, P$ we determine which frames to transmit for each of the video segments $(m, k),\ m = 1, \ldots, M$ and $k = 1, \ldots, K$. 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 Eq. 7 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 Eq. 7 is violated, we skip the virtual buffer and try to prefetch for the successor in the list, that is, we no longer prefetch 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 $p(m, k)$ values that are larger than or equal to 1. Note that this does not affect the ordering of the list. We found in our numerical experiments that it takes roughly 2.5 min to precompute the transmission schedule of length $P = 80,000$ frame periods for 70 video streams on a SUN ULTRA 170 workstation.

JSQ prefetching with prefetch delay. We now focus on the implementation issues that arise when adding the prefetch delay refinement to the JSQ policy. First of all, we see from Fig. 6 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 from 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 Fig. 8. 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 then starts 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 policy is that decrementing only the $p(m, k)$ values 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. The $p(b)$ value for virtual buffer b, however, is decremented by one, destroying the order of the linked list. The order must be restored before the JSQ algorithm starts over in the next slot. One approach to restoring 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 do not have to move the records around in the array but can sort indices into the array instead.

Lastly, note that the transmission schedule obtained with the JSQ prefetching policy with prefetch delay has a transient phase. This is because the second virtual buffers are initially in the "ignore" state for $N_k(m)$ frame periods, while they are in the "ignore" state for $N_k(m) - d_{\text{pre}}$ frame periods in all sub-

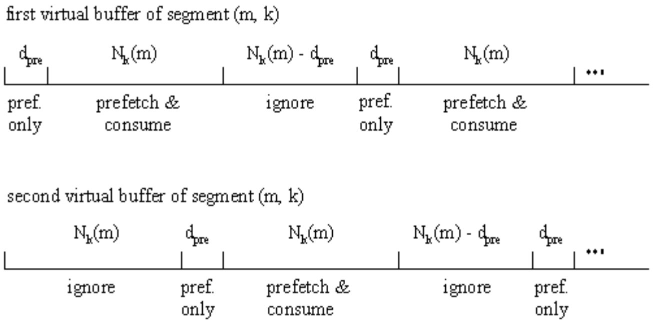

Fig. 8. Timing policy of state changes of the two virtual buffers of segment (m, k)

sequent cycles (Fig. 8). This transient behavior has to be taken into account when precomputing the transmission schedule. We found in our experiments that the transmission schedule typically reaches steady state rather quickly; typically after a few (3–5) warmup periods of length P frame periods. For precomputing the transmission schedule we run the JSQ algorithm with prefetch delay for a few warmup periods. We then record the transmission schedule by recording which frames are sent to each of the two virtual buffers of each of the MK ongoing streams over the next period.

NVOD with noncollaborative video smoothing. As noted at the end of Sect. 4.1, there is a large body of literature on smoothing techniques for clientcentric video streaming [8,12,19,25]. In this section of the appendix we briefly outline how these smoothing techniques can be employed in a NVOD system. In these smoothing techniques, the transmission schedule for a given video stream depends only on the traffic characteristics of that video stream. Thus, the transmission schedule of a video stream does not take into account the traffic characteristics of the other video streams. These schemes are therefore referred to as *noncollaborative*. JSQ prefetching, on the other hand, is *collaborative* as it takes all ongoing video streams into consideration. A detailed performance comparison of collaborative prefetching based on the JSQ principle and optimal smoothing [25], which is the optimal noncollaborative smoothing technique in that it minimizes peak rate and rate variability for a given smoothing delay budget, is reported in [22]. In summary, it was found that in typical scenarios the loss probability with JSQ prefetching is over two orders of magnitude smaller than when statistically multiplexing the optimally smoothed streams.

In this appendix we outline the implementation issues when using noncollaborative video smoothing for datacentric video streaming. First, consider smoothing techniques that introduce no startup delay. These techniques can be used to individually precompute the smoothed transmission schedule of all K segments of each video. The smoothed video segments are then broadcast periodically by multiplexing them onto the bufferless link, as discussed in Sect. 3.1. Note that smoothing without startup delay does not interfere with the continuity condition. Next, consider smoothing techniques with some startup delay. These techniques can again be used to individually precompute the smoothed segment schedules. To satisfy the continuity condition, the smoothed video segments are transmitted according to a timing policy that is akin to the policy depicted in Fig. 8. Note that the “prefetch only” phase of JSQ prefetching corresponds to the startup delay of the smoothing techniques. In both cases, smoothing with and without startup delay, the overall transmission schedule of the NVOD system is obtained by aggregating the MK individually precomputed segment schedules.

References

1. Aggarwal CC, Wolf JL, Yu PS (1996) A permutation-based pyramid broadcasting scheme for video-on-demand systems. In: Proceedings of the IEEE international conference on multimedia systems, Hiroshima, Japan, June 1966, pp 118–126

2. Aggarwal CC, Wolf JL, Yu PS (1996) On optimal batching policies for video-on-demand storage server. In: Proceedings of the IEEE international conference on multimedia systems, Hiroshima, Japan, June 1996, pp 253–258

3. Almeroth K, Ammar MH (1996) The use of multicast delivery to provide a scalable and interactive video-on-demand service. IEEE J Select Areas Commun 14(6):1110–1122

4. Birk Y, Mondri R (1999) Tailored transmissions for efficient Near-Video-On-Demand service. In: Proceedings of the international conference on multimedia computing and systems, Florence, Italy, June 1999, pp 226–231

5. Dalgic I, Tobagi FA (1996) 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, Palo Alto, CA, August 1996

6. Dan A, Sitaram D, Shahabuddin P (1994) Scheduling policies for an on-demand video server with batching. In: Proceedings of ACM Multimedia, San Francisco, October 1994, pp 15–23

7. Dan A, Sitaram D, Shahabuddin P (1996) Dynamic batching policies for an on-demand video server. Multimedia Sys 4(3):112–121

8. Feng W, Rexford J (1997) A comparison of bandwidth smoothing techniques for the transmission of prerecorded compressed video. In: Proceedings of IEEE INFOCOM, Kobe, Japan, April 1997, pp 58–67

9. Gao L, Kurose J, Towsley D (1998) Efficient schemes for broadcasting popular videos. In: Proceedings of NOSSDAV’98, Cambridge, UK, July 1998

10. Garret MW (1993) Contributions toward real-time services on packet networks. PhD thesis, Columbia University, New York, May 1993

11. Georgiadis L, Guerin R, Parekh AK (1994) Optimal multiplexing on a single link: delay and buffer requirements. In: Proceedings of IEEE INFOCOM’94, Toronto, June 1994, pp 524–532

12. Grossglauser M, Keshav S, Tse D (1995) RCBR: a simple and efficient service for multiple time-scale traffic. In: Proceedings

of ACM SIGCOMM, Cambridge, MA, August 1995, pp 219–230

13. Hua KA, Cai Y, Sheu S (1998) Exploiting client bandwidth for more efficient video broadcast. In: Proceedings of Computer Communications and Networks (IC3N'98), Lafayette, LA, October 1998

14. Hua KA, Sheu S (1997) Skyscraper broadcasting: a new broadcasting scheme for metropolitan video-on demand systems. In: Proceedings of ACM SIGCOMM, Cannes, France, September 1997, pp 89–100

15. Juhn L-S, Tseng LM (1997) Harmonic broadcasting for Video-on-Demand service. IEEE Trans Broadcast 43(3):268–271

16. Krunz M, Sass R, Hughes H (1995) Statistical characteristics and multiplexing of MPEG streams. In: Proceedings of IEEE INFOCOM, Boston, April 1995, pp 455–462

17. Lakshman TV, Ortega A, Reibman AR (1998) VBR video: tradeoffs and potentials. Proc IEEE 86(5):952–973

18. Li F, Nikolaidis I (1999) Trace-adaptive fragmentation for periodic broadcast of VBR video. In: Proceedings of the 9th international workshop on network and operating systems support for digital audio and video (NOSSDAV'99), Basking Ridge, NJ, June 1999, pp 253–264

19. McManus JM, Ross KW (1997) A dynamic programming methodology for managing prerecorded VBR sources in packet-switched networks. In: Proceedings of SPIE, Performance and control of network systems, Dallas, TX, November 1997, pp 140–154

20. Oh S, Huh Y, Konjevod G, Richa A, Reisslein M (2003) Collaborative prefetching of continuous media: a modular approach to achieve fairness and efficiency. Technical report, Deparment of Electrical Engeering, Arizona State University, Tempe, AZ, June 2003. available at `http://www.fulton.asu.edu/~mre`

21. Paris J-F, Carter SW, Long DDE (1998) Efficient broadcasting protocols for video on demand. In: Proceedings of the 6th international symposium on modeling, analysis and simulation of computer and telecommunication systems, Montreal, July 1998, pp 127–132

22. Reisslein M, Ross KW (1998) High-performance prefetching protocols for VBR prerecorded video. IEEE Netw 12(6):46–55

23. Reisslein M, Ross KW (2002) A framework for guaranteeing statistical QoS. IEEE/ACM Trans Netw 10(1):27–42

24. Rose O (1995) Statistical properties of MPEG video traffic and their impact on traffic modeling in ATM systems. Technical Report 101, University of Würzburg, Institute of Computer Science, Am Hubland, 97074 Würzburg, Germany, February 1995. ftp://ftp-info3.informatik.uni-wuerzburg.de/pub/MPEG/

25. Salehi J, Zhang Z, Kurose J, Towsley D (1997) Supporting stored video: reducing rate variability and end-to-end resource requirements through optimal smoothing. IEEE/ACM Trans Netw 6(4):397–410

26. Saparilla D, Ross KW, Reisslein M (1999) Periodic broadcasting with VBR-encoded video. In: Proceedings of IEEE INFOCOM, New York, March 1999, pp 464–471

27. Shi W, Ghandeharizadeh S (1997) Trading memory for disk bandwidth in video-on-demand servers. Technical report, University of Southern California, Los Angeles

28. Sincoski WD (1991) System architecture for a large scale video on demand service. Comput Netw ISDN Sys 22(2):155–162

29. Tan WS, Duong N, Princen J (1991) A comparison study of variable bit rate versus fixed bit rate video transmission. In: Proceedings of the Australian symposium on broadband switching and services, location, day month 1991, pp 134–141

30. Viswanathan S, Imielinski T (1996) Metropolitan area video-on-demand service using pyramid broadcasting. Multimedia Sys 4(4):197–208

31. Wang Y, Zhu Q (1998) Error control and concealment for video communication: a review. Proc IEEE 86(5):974–997

Martin Reisslein is an assistant professor in the Department of Electrical Engineering at Arizona State University, Tempe. He received the Dipl.-Ing. (FH) degree from the Fachhochschule Dieburg, Germany in 1994 and the M.S.E. degree from the University of Pennsylvania at Philadelphia in 1996, both in electrical engineering. He received his Ph.D. in systems engineering from the University of Pennsylvania in 1998. During the 1994–1995 academic year he visited the University of Pennsylvania as a Fulbright scholar. From July 1998 through October 2000 he was a scientist with the German National Research Center for Information Technology (GMD FOKUS), Berlin. While in Berlin he taught courses on performance evaluation and computer networking at the Technical University Berlin. He is editor-in-chief of the IEEE Communications Surveys and Tutorials and has served on the Technical Program Committees of IEEE Infocom, IEEE Globecom, and the IEEE International Symposium on Computer and Communications. He has organized sessions at the IEEE Computer Communications Workshop (CCW) and maintains an extensive library of video traces for network performance evaluation, including frame size traces of MPEG4- and H.263-encoded video, at `http://trace.eas.asu.edu`. He is corecipient of the Best Paper Award of the SPIE Photonics East 2000 – Terabit Optical Networking conference. His research interests are in the areas of Internet Quality of Service, video traffic characterization, wireless networking, and optical networking.

Despina Saparilla received her Ph.D. in systems engineering from the University of Pennsylvania at Philadelphia in 2000. As part of her Ph.D. research she joined the Multimedia Communications Department at Institut Eurecom, Sophia-Antipolis, France, where she worked on video streaming and video-on-demand technologies. Since January 2001, she has been working in the Content Networking Business Unit at Cisco Systems Inc., San Jose, CA on the design and development of multimedia streaming and content distribution products.

Keith W. Ross joined Polytechnic University as the Leonard Shustek Chair Professor of Computer Science in January 2003. Before joining Polytechnic University, he was a professor for 5 years in the Multimedia Communications Department at the Eurecom Institute in Sophia Antipolis, France. From 1985 through 1997 he was a professor in the Department of Systems Engineering at the University of Pennsylvania. He received his B.S.E.E from Tufts University, M.S.E.E. from Columbia University, and Ph.D. in computer and control engineering from The University of Michigan. Professor Ross has worked in stochastic modeling, QoS in packet-switched networks, video streaming, video on demand, multiservice loss networks, Web caching, content distribution networks, peer-to-peer networks, application-layer protocols, voiceover IP, optimization, queuing theory, optimal control of queues, and Markov decision processes. He is an associate editor for IEEE/ACM Transactions on Networking. Professor Ross is coauthor (with James F. Kurose) of the bestselling textbook *Computer Networking: A Top-Down Approach Featuring the Internet*, published by Addison-Wesley. He is also the author of the research monograph Multiservice Loss Models for Broadband Communication Networks, published by Springer. From July 1999 to July 2001, Professor Ross founded and led Wimba, an Internet startup that develops asynchronous voice products.