Efficient Support for Interactive Scanning Operations in MPEG-Based Video-On-Demand Systems*

Marwan Krunz† George Apostolopoulos‡

Abstract

In this paper, we present an efficient approach for supporting fast-scanning (FS) operations in MPEG-based video-on-demand (VOD) systems. This approach is based on storing multiple, differently encoded versions of the same movie at the server. A *normal version* is used for normal playback, while several *scan versions* are used for FS. Each scan version supports forward and backward FS at a given speedup. The server responds to a FS request by switching from the normal version to an appropriate scan version. Scanning versions are produced by encoding a sample of the raw frames using the same GOP pattern of the normal version. When a scanning version is decoded and played back at the normal frame rate, it gives a perceptual motion speedup. By being able to control the traffic envelopes of the scan versions, our approach can be integrated into a previously proposed framework for distributing archived, MPEG-coded video streams [25]. FS operations are supported using no or little extra network bandwidth beyond what is already allocated for normal playback. Mechanisms for controlling the traffic envelopes of the scan versions are presented. The actions taken by the server and the client's decoder in response to various types of interactive requests are described in detail. The latency incurred in implementing various interactive requests is shown to be within an acceptable range. Striping and disk scheduling strategies for storing various versions at the server are presented. Issues related to signalling and admission control are discussed.

keywords: MPEG, video scheduling, interactive video-on-demand, scanning operations.

1 Introduction

The maturing of video compression technologies, magnetic storage subsystems, and broadband networking has made video-on-demand (VOD) over computer networks more viable than ever. Major carriers have tested small-scale VOD systems, and companies that provide related services and products are emerging (cf. [22, 7]). To improve the marketability of VOD services and accelerate their wide-scale deployment, these services must support user interactivity at affordable cost. At minimum, an interactive VOD service must allow users to dynamically request basic VCR operations,

*Part of this paper was presented at the *ACM/SPIE Multimedia Computing and Networking Conference, 1998.*

†Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85718. Tel. (520) 621-8731. krunz@ece.arizona.edu. This work was partially supported by an NSF CAREER Award ANI-9733143 and partially by a University of Arizona Faculty-Grant Award.

‡Department of Computer Science, University of Maryland, College Park, MD 20742.

such as *stop-resume*, *pause-resume*, *slow motion*, *jump* (forward or backward), and *fast scanning* (i.e., viewing the movie forward or backward at multiple times the normal playback rate).

Interactivity is generally more difficult to provide in a VOD system than in a conventional VCR. This is attributed to the fact that in a typical VOD architecture, only part of the movie is in the client set-top box at a given time. While it is possible to prefetch the whole movie prior to its commencement, the excessive amount of memory needed inside the set-top box and the long prefetch time associated with transporting the movie at a reasonable amount of bandwidth preclude the viability of this approach. Consequently, in a typical VOD system the requested movie is *streamed* from the server to the client over a network. The implication of video streaming is that in addition to the client buffer size, the performance of the VOD system depends on network bandwidth and the round-trip delay between the server and the client. While these complications also exist in non-interactive (playback-only) VOD systems, they are particularly acute in the presence of interactivity for the following reasons: (1) the pattern of user interactivity is typically non-deterministic, and (2) certain interactive operations impose large bandwidth and processing demands on the VOD system.

The difficulty of supporting interactivity in a VOD system varies from one interactive function to another. A *stop*, *jump*, or *pause* followed by *resume* are relatively easy to support, as they do not require more bandwidth than what is required for normal playback. On the other hand, *fast-scanning* (FS) involves displaying frames at several times the normal rate. Transporting and decoding frames at multiple times the normal frame rate is prohibitively expensive and is infeasible with today's hardware decoders. Backward FS is even more difficult to support in compression schemes that involve motion interpolated frames, such as B frames in the MPEG scheme [21]. In the case of MPEG, all the reference frames in a Group of Pictures (GOP) must be decoded before B frames of that GOP can be played back in the reverse order.

Several approaches have been proposed to support interactivity in a VOD system. In [32] interactive operations, including scanning, are implemented at the client side using prefetched frames. The attractiveness of this approach lies in its transparency to both the network and the server. However, if a scanning operation lasts for an extended period of time, a significant portion of the movie must be prefetched. In addition to the large buffer requirement, excessive prefetching necessitates requesting the movie long before its commencement. Scanning operations can also be supported by transmitting frames at multiple times the normal frame rate over a communications channel that is different from the one used for normal playback [14, 35]. Since at a given point in time only a small percentage of users are in the interactive mode, the "interactive channel" can be shared by several users. However, in this case there is a small probability that a request for a FS operation will be rejected (i.e., FS operations are guaranteed on a statistical basis). During the scanning operation, video frames must be decoded at multiple times the normal decoding rate.

Interactivity has also been addressed in the context of batching [2, 1, 3, 13, 19, 29, 30, 41]. As an example, in [2] the authors assume that the VOD server operates in a multicast environment, whereby multiple instances of each movie are being simultaneously distributed. However, these instants have different logical times. VCR operations are implemented by moving the user to a multicast group with an appropriate logical time. Since the number of the different instances of the same movie is limited, this scheme can only support "discontinuous" VCR functions. The set-top buffer needed to support FS operations can be excessive if the number of multiple instances is small. Furthermore, the decoder is still required to process frames at multiple times the normal frame rate to achieve a FS effect.

FS functions can also be supported by dropping parts of the original compressed video stream [10, 11, 38, 9, 33, 31]. Dropping aims at reducing both the transport and the decoding requirements of FS without causing significant degradation in video quality. In MPEG-2, dropping is facilitated by various modes of scalability (spatial, temporal, and SNR) [21]. Spatial scalability, for example, provides the means to drop the less important data (the enhancement layer) and maintain the essential data (the base layer). Typically, dropping is performed *after* compression, so it must be done selectively to ensure that the dropped data will not result in significant degradation in video quality. For example, if whole MPEG frames are to be dropped, then dropping must take into account the dependency structure of the MPEG sequence. One possibility is to drop all B frames of an MPEG stream and transmit anchor frames (I and P) [9]. When the transmitted frames are played back at the normal playback rate, they give the visual perception of a FS. Another alternative is to drop MPEG frames on a per-GOP basis [11]. Since a GOP interval corresponds to about half of a second, this approach introduces discontinuities in the movie. A third approach is to skip a trailing part of each GOP and transmit the first few frames of the GOP [33] (the transmitted frames must be chosen in such a manner that they can be decoded independently of the skipped frames). The transmitted frames are then played back at the normal frame rate. A good discussion of these and other MPEG-related techniques is given in [38]. Instead of dropping frames after compression, some researchers suggested supporting FS operations using separate copies of the movie that are encoded at lower quality than the quality of the normal playback copy [37]. These "scan" copies include only I and P frames (i.e., B frames are not used). This makes it easier to provide backward fast-scan. The motion vectors of the predicted frames are encoded in such a manner so as to reduce the artifacts when playing frames in the reverse order.

In this work, we introduce an efficient approach for supporting *forward fast-scanning* (FFS) and *backward fast-scanning* (BFS) in a VOD system. Similar to [37], our approach is based on encoding separate copies of the movie to be used for FS operations, with each copy being generated by skipping raw video frames *before* compression. We refer to these versions as the scan versions, and to the one used for normal playback as the normal version. Each scan version is used to provide both BFS and FFS at a given speedup. Scan versions are encoded in a way such that when played back at the normal frame rate, they give the perception of a faster video in the forward or backward direction. In contrast to the approach in [37], the scan versions are encoded using the same GOP of the normal version (B frames are included). The encoding of a scan version is performed in a manner that enforces a particular *time-varying* traffic envelope for that version. This form of rate-controlled compression results in variable picture quality during FS. By making the envelopes of the scan versions identical or sufficiently close to the envelope of the normal version, FS can be integrated into a previously proposed framework for the distribution of archived, MPEG-coded video streams [25, 26]. In that framework, MPEG streams are characterized by time-varying deterministic envelopes. Using these envelopes, streams that are destined to a particular head-end switch are appropriately scheduled and statistically multiplexed at the server. Bandwidth is allocated based on the peak rate of the aggregate traffic, rather than the sum of the individual peak rates. Accordingly, each MPEG stream is guaranteed stringent and deterministic quality of service (constant frame rate) at bandwidth cost that is significantly less that the source peak rate. By generating scan versions that exhibit similar envelopes to the normal version, FS operations are transparent to the underlying network, and they can be supported with little or no extra bandwidth and at the same decoding rate of normal playback.

The paper is organized as follows. In Section 2 we briefly describe our previously proposed framework for video distribution based on time-varying traffic envelopes. In Section 3 the preprocessing steps required to support FS operations are presented. Section 4 provides detailed description of how FS-related interactivity is supported. Signalling between the client and the server is discussed in Section 5. In Section 6 we discuss the storage issue and we present striping strategies that are suitable for our FS approach. Admission control at the server is discussed in Section 7. Finally, Section 8 summarizes the paper and points to open research issues.

2 Envelope-Based Video Scheduling and Multiplexing

In this section, we give an overview of a previously proposed framework for the distribution of MPEG-coded video streams. Details of this framework and its related scheduling, multiplexing, resource allocation, and admission control aspects can be found in [25, 26, 40]. In this framework, a video distribution network consists of several fixed-capacity dedicated "bandwidth pipes" that extend from the server to remote head-end (HE) switches over a public network (Figure 1). These bandwidth pipes can be, for example, ATM virtual paths (VPs) onto which several video connections are multiplexed. We assume (as often the case) that the server and the HE switches are administered by one entity (i.e., the video-service provider), so that a VP can be terminated at a HE switch rather than at the end system. Clients request videos on demand by sending their requests to the server via one of the HE switches. Video streams are transported at a constant frame rate, but with per-stream bandwidth that is significantly less than the source peak rate. Since the frame transmission rate is the same as the playback rate, prefetching is not needed at the client set-top box. Bandwidth gain is achieved through statistical multiplexing of MPEG streams that are described by deterministic,

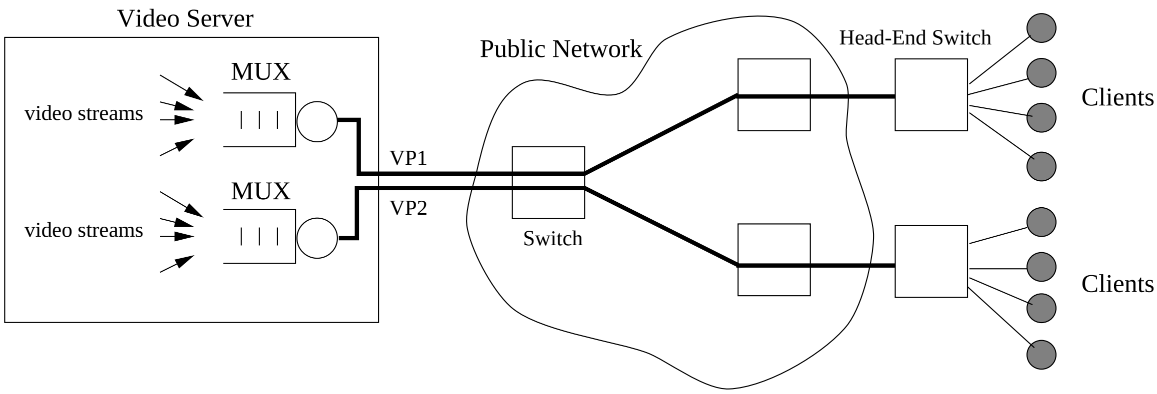


Figure 1: Video distribution network with two HE switches.

time-dependent traffic envelopes. An envelope here constitutes a time-varying upper bound on the bit rate. It is intended to capture the periodic structure of an MPEG stream (in terms of the repetition of the GOPs). The simplest form of our traffic envelope is called the *global envelope*, and is described as follows: For the ith MPEG stream, s_i, the global envelope is a periodic function (in time) that is parameterized by the 5-tuple $\left(I_{max}^{(i)},\ P_{max}^{(i)},\ B_{max}^{(i)},\ N^{(i)},\ M^{(i)}\right)$, where $I_{max}^{(i)}$ is the largest frame of s_i (typically, an I frame), $P_{max}^{(i)}$ is the largest P or B frame of s_i (typically, a P frame), and $B_{max}^{(i)}$ is the largest B frame of s_i. By construction, $I_{max}^{(i)} \geq P_{max}^{(i)} \geq B_{max}^{(i)}$. The remaining two parameters characterize the GOP pattern of the ith stream: $N^{(i)}$ is the length of a GOP (I-to-I frame distance) and $M^{(i)}$ is the P-to-P frame distance. An example of the global envelope is shown in Figure 2.

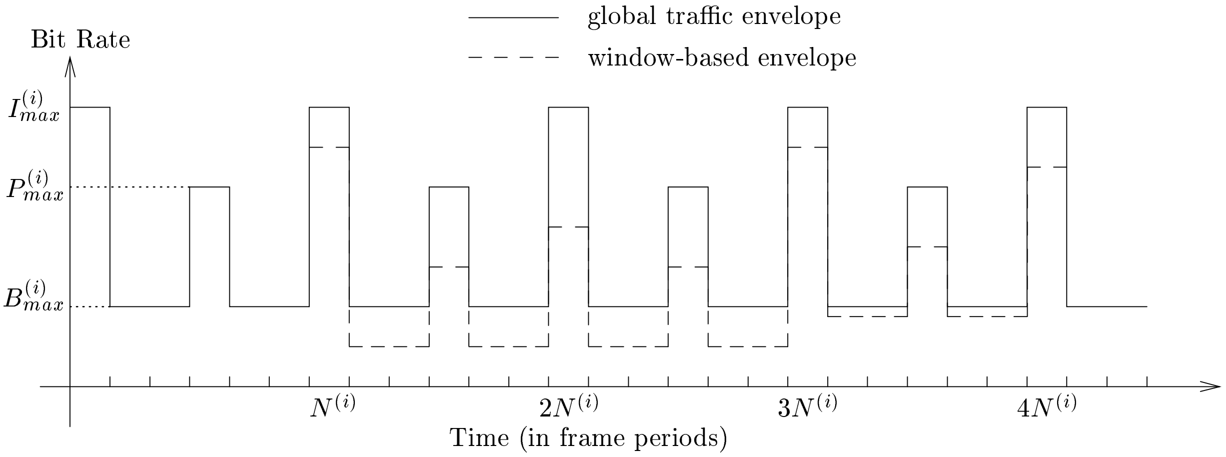


Figure 2: Example of global and window-based traffic envelopes ($N^{(i)} = 6$, $M^{(i)} = 3$).

Based on the global-envelope model, MPEG streams can be appropriately scheduled for multiplexing at the server. Consider n MPEG video streams, $s_1, \ldots, s_n$, that are destined to the same HE switch. Let $\overline{b}_i(t)$ be the traffic envelope of s_i, whose starting time is given by t_i. Let $\widetilde{N}$ be the least common multiple of $\{N^{(1)}, N^{(2)}, \ldots, N^{(n)}\}$. We define the *phase* of s_i by $u_i = t_i \bmod \widetilde{N}$, with $u_1 \triangleq 0$. In the special case when $N^{(i)} = N$ for all i, u_i describes the frame lag of a GOP of s_i relative to the closest GOP of s_1. The temporal relationships between the GOPs of the n streams are completely specified by $\boldsymbol{u} = (u_1, u_2, u_3, \ldots, u_n)$, which is referred to as the *arrangement*. Let $\overline{b}_{tot}(t)$ be the traffic envelope resulting from the superposition of the n streams; $\overline{b}_{tot}(t) = \sum_i \overline{b}_i(t - u_i)$. Note that $\overline{b}_{tot}(t)$ is periodic with period $\widetilde{N}$. The peak rate of $\overline{b}_{tot}(t)$ is given by

$$\max_{t \geq 0} \overline{b}_{tot}(t) = \max_{t \geq 0} \left(\sum_{i=1}^{n} \overline{b}_i(t - u_i) \right) \stackrel{\triangle}{=} nC(\boldsymbol{u}, n) \tag{1}$$

By allocating $nC(\boldsymbol{u}, n)$ of bandwidth to the aggregate traffic, each stream is guaranteed a constant-frame-rate delivery on an end-to-end basis. For most values of $\boldsymbol{u}$, $C(\boldsymbol{u}, n)$ (which is referred to as the *per-stream allocated bandwidth* (PSAB)) is smaller than the source peak rate. The allocated bandwidth $nC(\boldsymbol{u}, n)$ is updated dynamically upon the addition of a new video stream or the termination of an ongoing one. Stream scheduling is performed only for new video requests, and is done at the expense of delaying the service of a new request by no more than $\widetilde{N}$ frame periods.

An optimal scheduling policy is one that produces the *best* arrangement, $\boldsymbol{u}^*$, where

$$C(\boldsymbol{u}^*, n) \stackrel{\triangle}{=} \min_{u \in \mathcal{U}} C(\boldsymbol{u}, n) \tag{2}$$

and $\mathcal{U}$ is the set of all possible arrangements of n streams. In [25] optimal and suboptimal scheduling policies were proposed for homogeneous and heterogeneous multiplexed streams that are characterized by global envelopes. These policies resulted in PSAB of about 40-60% of the source peak rate (the actual value depends on the envelope). Bandwidth gain can be further improved using *window-based* traffic envelopes [40]. In this case, a pre-recorded MPEG sequence is divided into several segments that have the same number of frames. The time it takes to transmit one segment is called a *window*. As in the global-envelope model, five parameters are used to characterize the window-based envelope. However, in this case the values of the first three parameters (i.e., the maximum frame sizes) are computed for each segment of the movie (see Figure 2). Several efficient scheduling schemes were devised under window-based envelopes. Clearly, the smaller the window size the smaller the amount of allocated bandwidth, but the higher the computational complexity of updating the allocated bandwidth. For reasonable window sizes, the PSAB is about 15-30% of the source peak rate (depending on the envelope parameters and the window size). This bandwidth gain is comparable to the one achieved through video smoothing (e.g., [36, 32, 27, 15, 20]), with the advantages of (1) not requiring any buffer in the set-top box, (2) not depending on network delay variation, and (3) having a very small startup delay.

The framework in [25, 26] was originally designed for playback-only VOD, so it did not support client interactivity. Interactive operations other than FS can be easily integrated into that framework, and therefore will not be addressed further. In this paper, we focus on FS interactive operations and their integration into our VOD framework.

3 Preprocessing of Video Movies

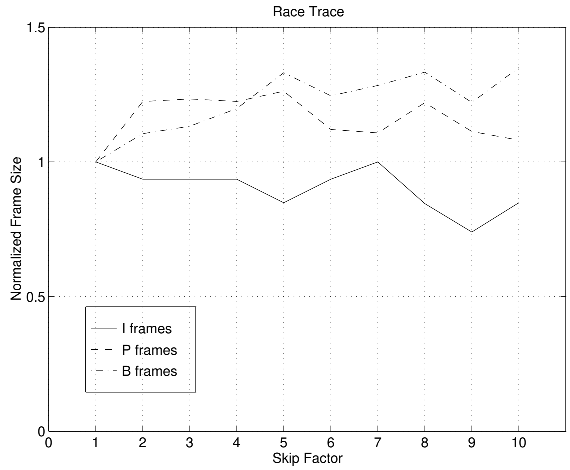
3.1 Scan Versions

To support FS operations, the server maintains multiple, differently encoded versions of each movie. One version, which is referred to as the *normal version*, is used for normal-speed playback. The other versions, which are referred to as the *scan versions*, are used for fast-scanning. Each scan version is used to support both FFS and BFS at a given speedup. The server switches between the various versions depending on the requested interactive operation (only one version is transmitted at a given instant of time). For a given speedup factor s ($s \geq 2$), the corresponding scan version is obtained by encoding a subset of the raw (i.e., uncompressed) frames of the original movie at a sampling rate of 1-to-s. By varying the sampling rate, any integer-valued speedup can be achieved. We refer to the sampling rate as the *skip factor*.

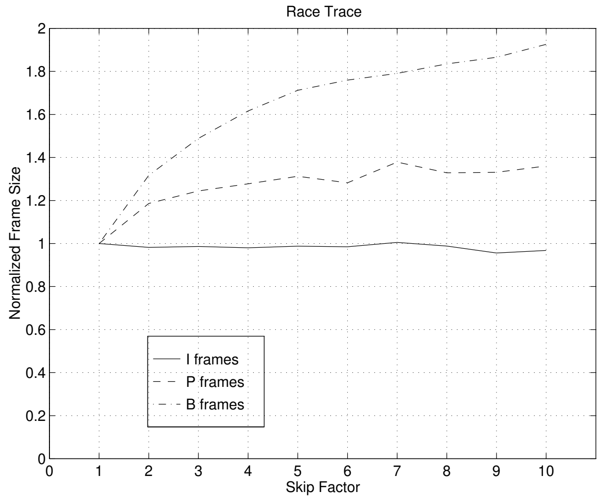
In order to support FS operations using the VOD framework discussed in the previous section, switching between the normal version and a scan version must be transparent to the network. For this purpose, scan versions are encoded using the *same* GOP pattern of the normal version and are transported at the normal frame rate. Furthermore, the traffic envelopes of the scan versions must be comparable in magnitude to the envelope of the normal version. If the scan versions are encoded using the same GOP pattern of the normal version, then it is easy to see that every raw frame that is encoded as an I frame in a scan version is also encoded as an I frame in the normal version (i.e., I frames of the scan versions constitute a subset of I frames of the normal version). To see that, consider an arbitrary raw frame that is coded as an I in a scan version. The index of this frame (i.e., its temporal order) is given by $1 + kN$, where N is the GOP length and k is some nonnegative integer. The index of the same raw frame in the normal version is $1 + skN$, which is an index of an I frame. That is, the same raw frame is encoded as an I frame in the normal version. Since I frames are intra-coded, their sizes are not affected by other frames. Thus, every I frame of the scan version must exist in the normal version (but not the opposite).

Accordingly, I_{max} in the global envelope of a scan version is less than or equal to I_{max} in the global envelope of the normal version. This is not the case for P and B frames, whose sizes will be affected by frame skipping. Both types involve motion compensation (prediction or interpolation), which exploits the similarities between consecutive frames to reduce the frame size. Frame skipping increases the differences between successive images, resulting in larger P and B frames. The impact of frame skipping on the maximum and average frame sizes is illustrated in Figure 3 for the *Race* clip. Before skipping, this clip consists of 1000 frames with frame dimensions of 320×240 pixels. For each skip factor, encoding was performed using an MPEG-2 software encoder with $N = 6$ and $M = 3$. The quantization values were set to 8, 10, and 25, for I, P, and B frames, respectively.

From this figure, it is clear that I_{max} of a scan version is always bounded by I_{max} of the normal version. In contrast, both P_{max} and B_{max} tend to increase with the skip factor, though not necessarily



(a) Maximum frame sizes.



(b) Average frame sizes.

Figure 3: Frame sizes of a scan version versus the skip factor (values are normalized with respect to their counterparts in the normal version).

in a monotone fashion. The nonmonotonic behavior is attributed to the fact that a large frame (I, P, or B) of a given scan version may be missed by another scan version that has a higher skip factor. Part (b) of the figure shows that the average size of I frames is almost unaffected by frame skipping. In contrast, the average sizes of P and B frames tend to increase with the skip factor.

3.2 Controlling the Envelopes of the Scan Versions

As indicated in Figure 3(a), encoding a sample of the raw frames may result in higher values of P_{max} and B_{max}. To generate scan versions with comparable envelopes to that of the normal version, the encoding of P and B frames of a scan version must be rate controlled. A common approach to control the size of an MPEG frame is to vary the quantization factor (the larger the quantization factor, the smaller the frame size). Varying the quantization factors on a per-frame basis results in variable video quality during FS (however, video quality is still constant during normal playback).

Without loss of generality, we consider the case when the envelopes are global. Extension to window-based envelopes is straightforward. To bound the sizes of P and B frames of a scan version, the encoder uses two predefined upper thresholds $T_P^{(u)}$ and $T_B^{(u)}$:

$$T_P^{(u)} \triangleq P_{max}(1 + S_P^{(u)}) \tag{3}$$
$$T_B^{(u)} \triangleq B_{max}(1 + S_B^{(u)}) \tag{4}$$

where P_{max} and B_{max} are for the normal version, and $S_P^{(u)}$ and $S_B^{(u)}$ are nonnegative constants. A P (B) frame in a scan version is encoded such that its size is no greater than $T_P^{(u)}$ ($T_B^{(u)}$). When $S_P^{(u)}$ or $S_B^{(u)}$ is positive, the envelope of a scan version is allowed to exceed the envelope of the normal version by no more than a fixed amount. In the case of window-based envelopes, $T_P^{(u)}$ ($T_B^{(u)}$) varies from one window to another, depending on the variations in P_{max} (B_{max}).

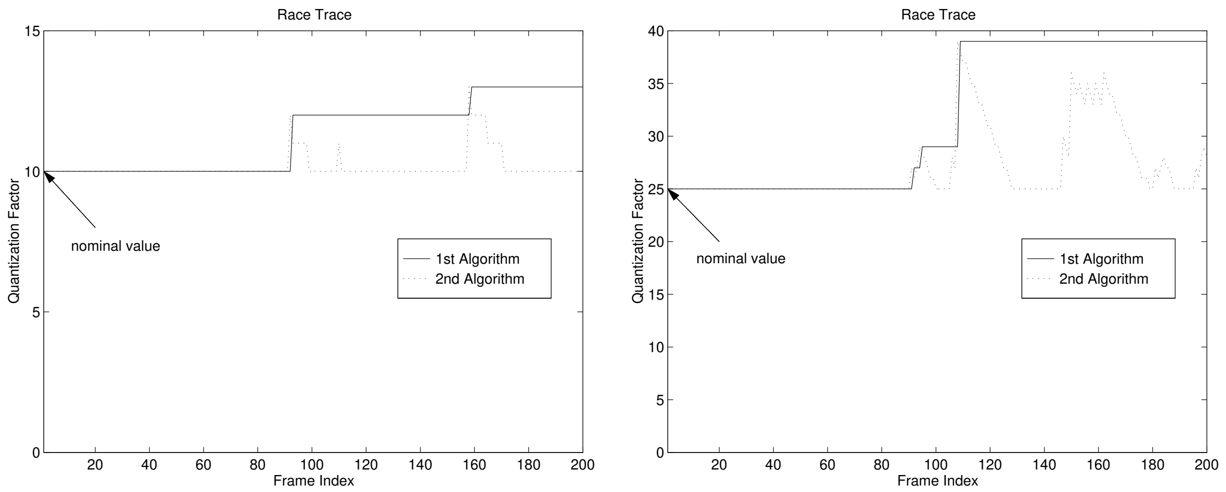
After a raw frame of a scan version has been encoded as a P or a B frame, the encoding algorithm checks whether the size of the compressed frame is below the associated upper threshold. If it is not, then the quantization factor for the corresponding frame type is increased by one and the raw frame is re-encoded. This procedure is repeated until the size of the compressed frame is smaller than the corresponding upper threshold.

Two different approaches can be used to initialize the quantization value when a new P or B frame is to be encoded. In the first approach (or algorithm), when a frame is to be encoded for the first time, the encoder starts with the last quantization value that was used in the encoding of the previous frame of the same type (i.e., the encoder does not reset the quantization value). The main problem with this approach is that the quantization value might be kept unnecessarily high following the encoding of a very large frame, resulting in an unnecessarily low quality during FS. While it is important to produce scan versions with comparable envelopes to that of the normal version (for scheduling and bandwidth allocation purposes), there is no advantage in reducing these envelopes below the envelope of the normal version.

In the second approach, the encoding algorithm tries to track the nominal quantization value of the corresponding frame type (the nominal quantization values are the ones used in the encoding of the normal version). Consider the encoding of a P frame (similar discussion applies to B frames). In the first encoding attempt, the encoder checks the final quantization value that was used to encode the previous P frame. If that value is equal to the nominal quantization value for P frames, then it is taken as the initial quantization value for the current frame. If on the other hand, the last quantization value of the previous P frame is larger than the nominal value, then the quantization value for the current frame is initialized to the last quantization value *minus* one. After the first encoding attempt, if the resulting frame size is within the upper bound, the encoder proceeds to the next frame. Otherwise, the quantization value is incremented and the same raw frame is re-encoded, as in the first approach. The advantage of the second approach is that it tries to produce a FS effect with the same constant quality of normal playback, but when this is not possible it minimizes the *fluctuation* in video quality during FS.

Figure 4 depicts the variations in the quantization values for P and B frames when $S_P^{(u)} = S_B^{(u)} = 0.05$ and $s = 5$. In these experiments, the nominal quantization values for P and B frames are 10 and 25, respectively. Note that the quantization factors for type-P and type-B frames are plotted versus the index of *every* frame in the scan version (including the indices of I frames).

In the second encoding approach, video quality during FS varies smoothly around the nominal quality at the expense of an increase in the number of encoding attempts. Since encoding in VOD is done off-line, the encoding time may be less important than video quality. In any case, a tradeoff can always be maintained. Table 1 compares the two encoding approaches when applied to the *Race* clip



(a) P frames. (b) B frames.

Figure 4: Variations in the quantization values during the encoding of a scan version ($s = 5$).

at $s = 5$. The performance is measured in terms of the average "lost" bandwidth and the ratio of the number of re-encoding attempts to the total number of frames. We define the average lost bandwidth as the ratio of the difference between the mean values of the traffic envelopes of the normal and scan versions to the mean value of the traffic envelope of the normal version.

Upper Thresholds		% of Average Bandwidth Loss		% of Re-encoding Attempts	
$S_P^{(u)}$	$S_B^{(u)}$	1st Approach	2nd Approach	1st Approach	2nd Approach
0	0	26.7%	22.5%	8.5%	30.0%
0	0.05	25.8%	21.8%	7.0%	21.5%
0.1	0.1	24.3%	20.9%	5.0%	14.5%
0.1	0.2	22.6%	20.7%	2.5%	9%
0.2	0.3	21.0%	19.8%	1.5%	4%

Table 1: Comparison between two alternative approaches for encoding the scan versions ($s = 5$).

The two approaches can be contrasted with respect to video quality using the peak signal-to-noise ratio (PSNR). PSNR measures the difference between two images by comparing their respective pixel values. To contrast our encoding algorithms, we use the PSNR of the Y-component of a decoded frame. The PSNR is obtained by comparing a decoded frame that was encoded using one of the two algorithms to the original raw frame. Figure 5 depicts the resulting PSNR values for *Race* movie with $s = 5$ and $S_P^{(u)} = S_B^{(u)} = 0.05$. Both approaches achieve acceptable quality (PSNR is sufficiently large). The average PSNR value for the 200 frames is 36.9 dB for the first algorithm and 37.5 dB for the second, i.e., the average quality is slightly better under the second algorithm. The absolute values of the PSNR do not convey the advantage of the second encoding approach. For this purpose, we compute the PSNR values for the 200 frames when encoding is done without any constraints (i.e., no upper bounds are imposed), and use these values as a reference. For each frame, we compute the difference between its reference PSNR value and the PSNR value resulting from each of the two rate-control encoding algorithms. These differences are plotted in Figure 6 for a segment of the scan version. In this figure, a large value indicates a large deviation from the reference PSNR, and thus lower quality. Clearly, the second algorithm achieves better quality than the first approach, but at the expense of more encoding attempts.

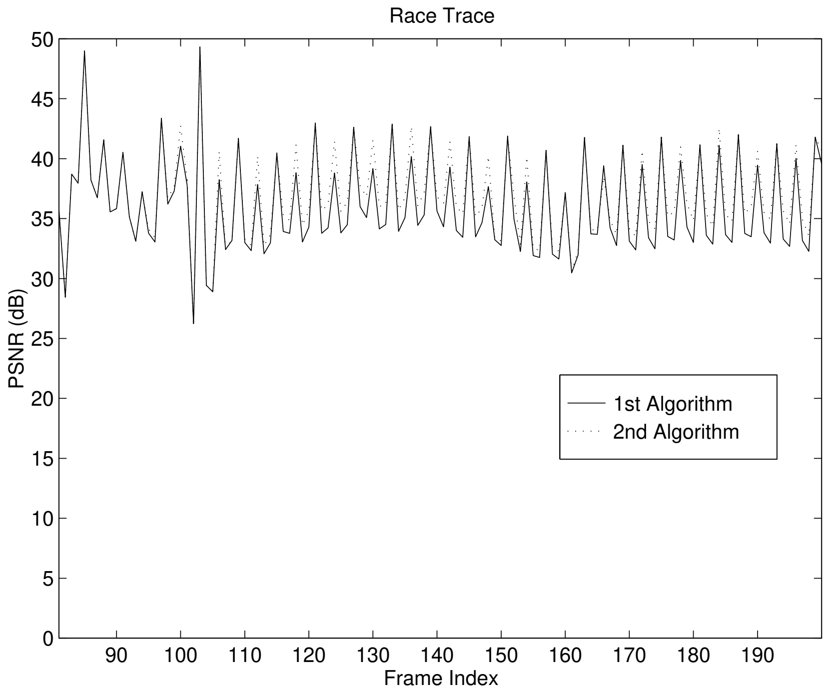


Figure 5: PSNR for the encoded frames of a scan version.

3.3 Storage Overhead

Generating separate copies for FS operations comes at the expense of extra storage at the server. To evaluate the storage overhead, we take into account the following considerations:

1. As explained in Section 4.4, BFS and FFS with the same speedup can be supported using the same scan version.

2. I frames of the scan versions need not be stored, since they constitute a subset of I frames of the normal version.

3. The number of encoded frames in a scan version is inversely proportional to the skip factor.

Given the above considerations, the storage overhead of the scan versions can be computed as follows. Without loss of generality, we consider the case of global traffic envelopes. Let f be the number of frames in the normal version. Of these frames, f/N are I frames, $f(1/M - 1/N)$ are P

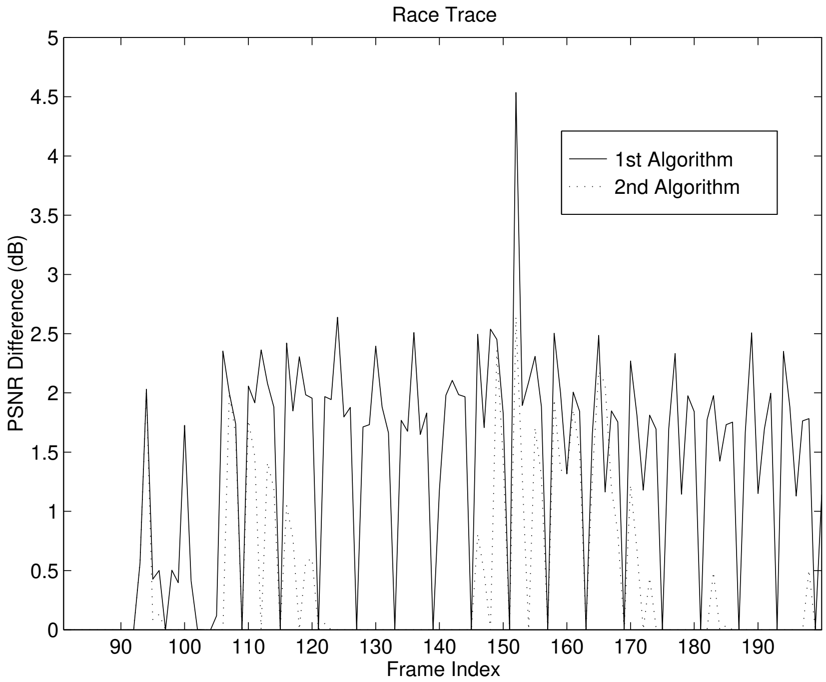


Figure 6: Difference between PSNR under unconstrained encoding and PSNR under constrained encoding.

frames, and $f(1-1/M)$ are B frames. The storage requirement of the normal version is given by

$$W_{norm} = fI_{avg}/N + fP_{avg}(1/M - 1/N) + fB_{avg}(1 - 1/M) \tag{5}$$

where I_{avg}, P_{avg} and B_{avg} are the average frame sizes of I, P, and B frames in the normal version. Let $P_{avg}(s)$ and $B_{avg}(s)$ be the average sizes of P and B frames in a scan version with skip factor s. Then, the storage requirement of this scan version is given by

$$W_{scan}(s) = f(1/M - 1/N)P_{avg}(s)/s + f(1 - 1/M)B_{avg}(s)/s \tag{6}$$

For n scan versions with skip factors $s_1, \ldots, s_n$, the relative increase in the storage requirement is given by $\sum_i W_{scan}(s_i)/W_{norm}$.

Numerical examples that show the relative increase in the storage requirement are given in Figures 7-9 for the *Race* clip. Figure 7 depicts the relative storage overhead of a scan version as a function of s when different upper thresholds are used. The upper threshold has a negligible impact on the storage overhead. For $s \geq 4$, the storage overhead of a scan version is no more than 25% of the storage requirement of the normal version. Figure 8 shows the increase in storage as a function of the GOP length (N) with $M = 3$ and $S_P^{(u)} = S_B^{(u)} = 0$. The storage overhead increases slowly with N. In practice, a VOD system may support two or more skip factors. For two skip factors,

Figure 9 depicts the total increase in storage when s_1 is fixed and s_2 is varied. For reasonable values of s_1 and s_2, the increase in storage is below 60%.

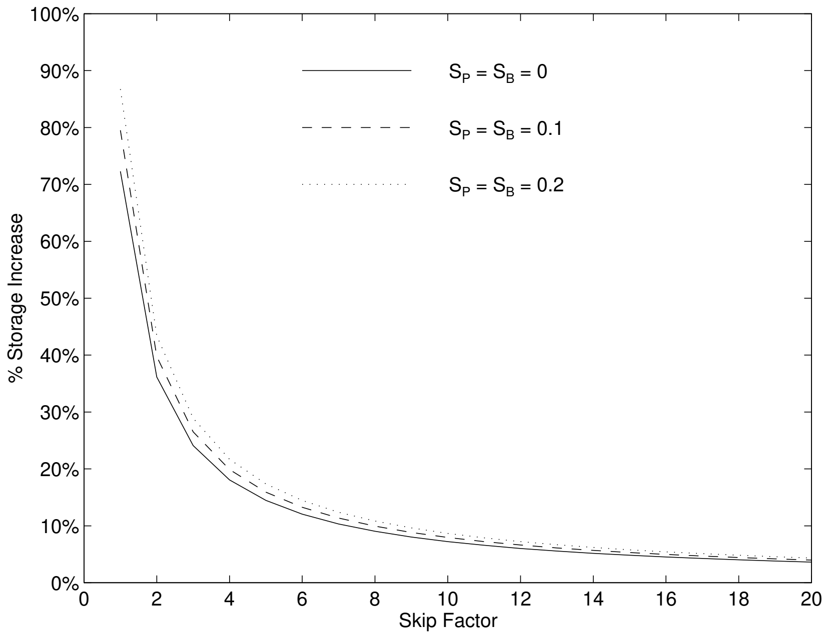


Figure 7: Relative increase in storage as a function of the skip factor ($N = 15$, $M = 3$).

4 Switching Between Normal and Scan Versions

In this section, we describe how switching between versions is used to support various FS-related operations. The notation that we use to specify a frame consists of a letter for the frame type and a number that indicates the logical time of that frame (i.e., the time relative to the events in the movie). This convention applies to all versions. Thus, `B16` in a scan version is a B frame that is obtained by encoding the 16th raw frame of the original movie. This B frame is *not* necessarily the 16th frame in the temporal order of that scan version (for example, if $s = 2$ then B16 is the 8th frame in the temporal order of the scan version).

Before discussing our solutions to FS operations, we first describe the typical operation of an MPEG decoder during normal playback.

4.1 Operation During Normal Playback

Because of the interpolative nature of B compression, the decoding of a B frame depends on two reference frames (I or P), both of which must be transmitted and decoded before the B frame is decoded. One of these reference frames comes after the B frame in the temporal order. To enable continuous playback at the receiver, MPEG frames are transmitted over the network according to

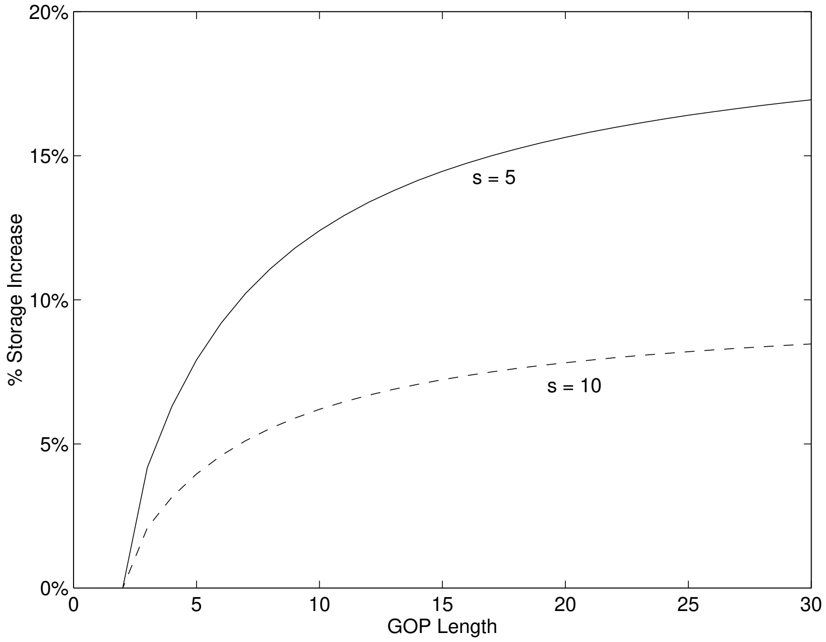


Figure 8: Relative increase in storage as a function of the GOP length ($M = 3$, $S_P^{(u)} = S_B^{(u)} = 0$).

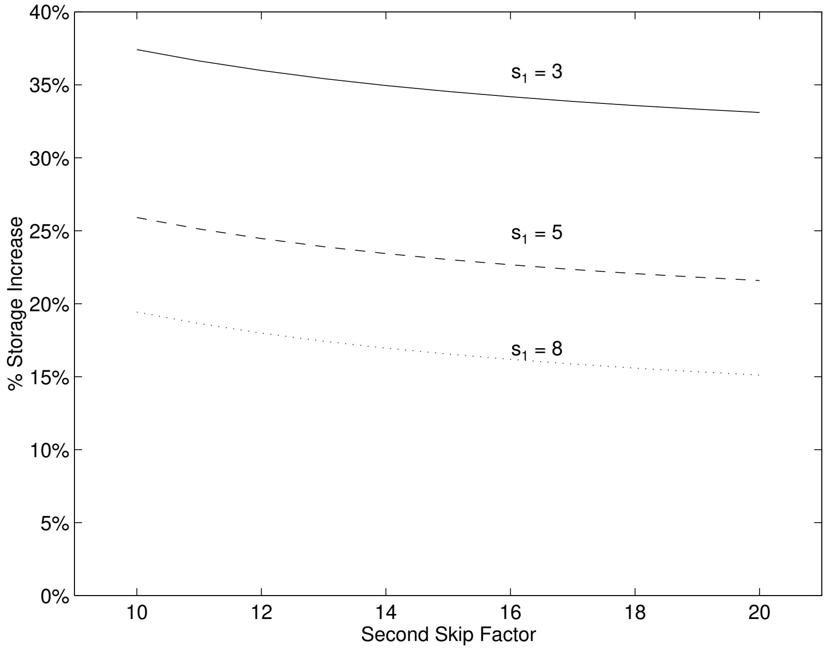


Figure 9: Relative increase in storage as a function of the second skip factor s_2 ($N = 15$, $M = 3$, $S_P^{(u)} = S_B^{(u)} = 0$).

their decoding order. Thus, the transmission (and decoding) order of an MPEG sequence is different from its temporal (playback) order. An example of the temporal and transmission orders of a normal version is shown in Figure 10. In order to decode frames B2 and B3, both I1 and P4 must be first transmitted and decoded. Thus, P4 is sent before B2 and B3. The process of transmitting, decoding, and displaying frames proceeds as follows. Starting at time $t = 0$ (the time unit is taken as one frame period), the server transmits frames according to their transmission order. Ignoring network delays for the time being, the decoder receives and decodes frame I1 during the time interval $[0, 1)$. It maintains an uncompressed copy of this frame to be used in decoding subsequent frames. During the interval $[1, 2)$, frame P4 is received, decoded, and stored in the frame buffer (note that P4 is decoded with reference to the uncompressed version of I1). Playback starts with I1, which is displayed in the interval $[2, 3)$ (two time units after it was received). During the same interval, B2 is received and decoded. During the interval $[3, 4)$, B2 is displayed, and B3 is received and decoded. During the interval $[4, 5)$, B3 is displayed, and I7 is received, decoded, and stored in one of the two frame buffers (at this point the decoder discards I1). In the subsequent interval, P4 (which has already been received and decoded) is displayed, and B5 is received and decoded using the uncompressed P4 and I7 frames that are in the frame buffer. And so on. In the above discussion, we have assumed that a frame is received and decoded within one time unit, i.e., the decoder is capable of decoding 30 frames per second (assuming an NTSC format). Current hardware MPEG decoders are close to achieving this performance.

I1 B2 B3 P4 B5 B6 I7 B8 B9 P10 B11 B12 I13 $\cdots$

(a) Temporal order

I1 P4 B2 B3 I7 B5 B6 P10 B8 B9 I13 B11 B12 $\cdots$

(b) Transmission order

Figure 10: Temporal and transmission orders of a normal version ($N = 6$, $M = 3$).

The decoder maintains a two-frame buffer, which contains the two most recently decoded reference frames. An incoming P frame is decoded with reference to the most recent of these two, while an incoming B frame is decoded with reference to both of them.

4.2 Switching From Normal Playback to FFS

Interactive FS operations are implemented at the server by switching between the normal version and one or more scan versions. Switching from one version to another is performed on-line, in response to a client request. In this section, we describe how switching is used to implement FFS.

Similar to the situation during normal playback, the frames of the scan version have a different transmission order than their temporal order (in the case of FFS, the temporal order of a scan version is the same as its playback order). Figure 11 depicts the temporal and transmission orders of a scan version with $s = 2$. Two successive I frames in a scan version differ in their *logical times* by sN frame periods (the logical time is the time relative to the events in the movie).

I1 B3 B5 P7 B9 B11 I13 B15 B17 P19 B21 B23 I25 ⋯

(a) Temporal order

I1 P7 B3 B5 I13 B9 B11 P19 B15 B17 I25 B21 B23 ⋯

(b) Transmission order

Figure 11: Temporal and transmission orders of a scan version ($N = 6$, $M = 3$, $s = 2$).

Several considerations must be taken into account when switching between versions at the server. First, the periodic structure of the GOP must be maintained for the whole transmitted sequence, including the transitions from one version to another. Second, the received frames must be decoded with minimal visual artifacts. Third, it is desirable that switching between versions be transparent to the normal operation of the decoder. Fourth, the amount of time that a client has to wait for before the FS operation is initiated must be as small as possible. For switching from normal playback to FFS and vice versa, the approach presented below satisfies all of these considerations.

To maintain the GOP periodicity, switching from a normal to a scan version must take place at an I frame. Furthermore, to enable correct decoding of all P and B frames of both normal and scan versions, this I frame must be common to both versions. Consider first switching from normal playback to FFS. When the FFS request arrives at the server, the server continues to send frames from the normal version up to (and excluding) the first P frame that follows a common I frame. From that point and on, the server switches to the scan version. The following two examples explain the switching process. Both examples use the normal and scan versions of Figures 10 and 11, respectively. In Figure 12(a) the FFS request arrives at the server after I13 (a common I frame), B11, and B12 have been sent, but before P16 is sent (which is the first P frame to follow I13 in the normal transmission order). B12 is received and decoded during the time slot in which B11 is played back. After transmitting B12 of the normal version, the server switches to the scan version, starting with frame P19. Meanwhile, the receiver decodes and plays back frames B12 and I13 of the normal version. It receives and decodes P19 of the scan version during the playback of B12. When P19 is to be decoded, the frame buffer at the decoder contains the decoded I13 and P10. Thus, P19 is decoded properly with reference to I13. Then, the decoded P19 replaces I13 in the frame buffer, and the decoded I13 replaces P10. When B15 of the scan version is received, it is decoded properly using I13 and P19. The switching operation is transparent to the decoder.

A second example is shown in Figure 12(b), where the FFS request arrives at the server after P16 of the normal version has just been transmitted. In this case, the server continues to send frames from the normal version up to (and including) B24. This essentially corresponds to continuing to play back frames from the normal version until the next common I frame (I25). After that, the server switches to the scan version, starting with P31, B27, B29, etc. Frame P31 is decoded using I25, which is common to both versions. This example gives the worst-case latency, in which the receiver continues normal playback sN frame periods from the time the FFS request is issued. Assuming that each GOP of the normal version corresponds to half of a second, the worst-case latency is $s/2$ seconds (the average latency is $s/4$ seconds). In requesting FFS, the client is trying to advance fast in the movie. Thus, extending normal playback by few seconds prior to initiating FFS is acceptable. Note that there is no disruption in the playback during the transition from normal to FFS.

	⊢ normal version ⊣			⊢		scan version			⊣
Received & Decoded	I13	B11	B12	P19	B15	B17	I25	B21	⋯
Displayed	B9	P10	B11	B12	I13	B15	B17	P19	⋯
Time	9	10	11	12	13	14	15	16	⋯
			↑		⟼				
			FFS request		start FFS				

(a) Example 1

	⊢				normal version					⊣	⊢ scan version ⊣		
Received & Decoded	I13	B11	B12	P16	B14	B15	⋯	I25	B23	B24	P31	B27	B29
Displayed	B9	P10	B11	B12	I13	B14	⋯	B21	P22	B23	B24	I25	B27
Time	9	10	11	12	13	14	⋯	21	22	23	24	25	26
				↑				(normal playback)					⟼
				FFS request								start FFS	

(b) Example 2

Figure 12: Switching from normal playback to FFS.

4.3 Switching From FFS to Normal Playback

We present two different approaches for switching from FFS to normal playback. The first approach is very similar to the one used to switch from normal playback to FFS. Upon receiving a request for normal playback, the server continues to send frames from the scan version until the next common I frame (it also sends the B frames of the scan version that precedes this I frame in the temporal order, but comes after it in the transmission order). After that, the server switches to the normal version. This approach is illustrated in the example in Figure 13 ($N = 6$, $M = 3$, $s = 2$). Here, the movie is in a FFS mode when the client requests normal playback during the display of frame B23. At that point, the server has just (or in the process of) transmitting P31. Ideally, the receiver should proceed by playing back frames with indices 24, 25, 26, etc. To decode B24 the decoder needs P22 of the normal version. However, if the server transmits P22, this frame will eventually be played back, causing undesirable artifacts. Neither can the server start from P28 of the normal version, since the decoder will incorrectly decode this frame with reference to P31 of the scan version. To ensure that switching to normal playback is done with all received frames being decoded properly, with no artifacts, and without any modifications to the normal operation of the decoder, the server must continue to send frames from the scan version until the next common I frame (I37). After that, it switches to the normal version, starting with P40, which can be decoded properly using I37. This means that FFS will be extended beyond the point at which normal playback was requested. In the worst case, normal playback is resumed at a logical time that is $s/2$ seconds from the logical time of the normal-to-FFS request (since N frames of the scan version correspond to $sN/2N = s/2$ seconds worth of video). However, it takes a maximum of only 1/2 second of *real time* to reach the appropriate switching point.

	⊢			scan version					⊣	⊢	normal version ⊣		
Received & Decoded	I25	B21	B23	P31	B27	B29	I37	B33	B35	P40	B38	B39	I43
Displayed	B17	P19	B21	B23	I25	B27	B29	P31	B33	B35	I37	B38	B39
		(FFS)		↑		(continue FFS)					⟼		
		normal playback request									start normal playback		

Figure 13: First approach for switching from FFS to normal playback.

The above approach has the advantage of being transparent to the decoder (i.e., the decoder need not know that a switch from a scan version to the normal version has occurred). However, it has the disadvantage that normal playback resumes at a later point than requested, with a worst-case difference of $s/2$ seconds of movie time. This difference can be reduced using a second switching approach, which will be described with reference to the examples in Figure 14 ($N = 6,\ M = 3,\ s = 2$). In this approach, normal playback resumes from the subsequent I frame of the normal version, which is not necessarily a common I frame. Consider Figure 14(a). Suppose during the display of B23 of the scan version, the client requests normal playback. The server has just sent frame P31 of the scan version. The first I frame of the normal version that should be played back is I25 (in this particular example, I25 happened to be common to both versions). The client continues FFS until I25. To resume normal playback from that point, the server must send P28, B26, and B27 of the normal version. However, since the server has just transmitted a P frame (P31), it must wait for up to N frame periods before transmitting P28 (this is needed to maintain the GOP periodicity of the transmitted sequence). While waiting for P28, the client continues to display I25 for a maximum pause period of $N/2N = 1/2$ second. During the pause period, the decoder ignores any frames sent by the server (in Figure 14 such frames are represented by 'X' for 'don't care'). Of course, a mechanism is needed to inform the decoder when to start accepting and decoding incoming frames. Such a mechanism will be described in a later section. Note that this switching approach is *not* transparent to the decoder.

A second example is shown in Figure 14(b), where normal playback resumes from a non-common

I frame (I31). FFS is continued until the frame of the scan version with the closest logical time to I31. In our example, this frame is P31. Frame P31 is never displayed, but is only used to decode B27 and B29 of the scan version. After receiving and displaying I31 of the normal version, the movie pauses at that frame until P34 of the normal version has been received and decoded (since two reference frames are needed to stream the playback process). It can be shown that the logical time of the last displayed frame from the scan version is no farther than s frame periods from the logical time of the subsequent I frame of the normal version. Thus, in the transition from the scan version to the normal version, the speedup of the motion picture is somewhere between normal playback and FFS (in our example, B29 of the scan version was followed by I31 of the normal version, and the transition appears as a continuation of FFS).

In the second FFS-to-normal switching approach, normal playback resumes at a logical time that is no farther than 1/2 second (N frame periods) from the logical time of the FFS-to-normal request. This is compared to $s/2$ seconds in the first approach. The maximum time that a client has to wait for before normal playback is resumed is $N/s + M$ frame periods; N/s frame periods of extended FFS and M frame periods of pause. This amounts to $1/2s + M/2N$ seconds, which is less than one second.

	$\vdash$			scan version					$\dashv$	$\vdash$	normal version $\dashv$		
Received & Decoded	I25	B21	B23	P31	X	X	X	X	X	P28	B26	B27	I31
Displayed	B17	P19	B21	B23	I25	I25	I25	I25	I25	I25	I25	B26	B27
		(FFS)		$\uparrow$			(pause)				$\longmapsto$		
		normal playback request									start normal playback		

(a) Example 1

	$\vdash$ scan version $\dashv$			$\vdash$			normal version				$\dashv$
Received & Decoded	P31	B27	B29	I31	X	X	P34	B32	B33	I37	B35
Displayed	B23	I25	B27	B29	I31	I31	I31	I31	B32	B33	P34
	(FFS)		$\uparrow$			(pause)			$\longmapsto$		
		normal playback request							start normal playback		

(b) Example 2

Figure 14: Second approach for switching from FFS to normal playback.

4.4 Switching From Normal Playback to BFS

Instead of generating a scan version to be used exclusively for BFS, we use one scan version for both FFS and BFS at a given speedup. However, in this case the dependency structure of an MPEG sequence must be taken into account when transmitting frames during BFS and when decoding and displaying these frames at the client side. The complications associated with the inter-frame dependencies of an MPEG sequence are illustrated in Figure 15 for a normal version with $N = 9$ and $M = 3$. For simplicity, we consider a BFS operation with $s = 1$ (i.e., the scan version is the same as the normal version). BFS is requested during the playback of frame B18. To initiate backward playback starting from B18, the decoder needs reference frames P16 and I19, but only I19 is available in the frame buffer (during the playback of B18, P22 is decoded and stored in the frame buffer, pushing out the previously decoded P16). Frame P16 depends on P13, which in turn depends on I10. Thus, to initiate backward playback from B18, frames I10, P13, and P16 must first be retransmitted and decoded.

Received & Decoded	···	B9	P13	B11	B12	P16	B14	B15	I19	B17	B18	P22
Displayed	···	B8	B9	I10	B11	B12	P13	B14	B15	P16	B17	B18
					(normal playback)							↑
												BFS request

Figure 15: Problems associated with reversing the playback direction.

In addressing the direction reversal problem, we first consider switching from normal playback to backward playback, which is a special case of BFS with $s = 1$. We then consider BFS for any speedup s.

4.4.1 Normal Playback to Backward Playback

To implement backward playback (BPB), the server uses a different transmission order than the one used during (forward) normal playback. After receiving a BPB request, the server initiates the BPB operation starting from the subsequent reference frame (I or P) of the current GOP. Before initiating BPB, the decoder must receive and decode the reference frames of the current *and* previous GOPs. Consider the situation in Figure 16. The temporal order of the underlying MPEG sequence is shown in Part (a) of the figure. Suppose that a BPB is issued during the playback of B36. The subsequent reference frame that follows B36 in the temporal order is I37. Thus, the client continues normal playback until the display of I37. Meanwhile, the server continues sending the frames of the normal version that are needed to maintain normal playback until (and including) I37. After that, the server starts sending the reference frames of the current and previous GOPs. A maximum of $2N/M$ reference frames need to be decoded and stored in the frame buffer before BPB is initiated. In our example, the client had already decoded and stored I37 when the BPB was issued. Hence, before initiating BPB the decoder must receive and decode I28, P31, and P34 of the current GOP as well as I19 and P22 of the previous GOP. While these frames are being transmitted and decoded, the movie pauses for a maximum duration of two GOPs (one second). To minimize the pause duration, the following guidelines are followed *whenever possible*: (1) reference frames of the present GOP are sent before reference frames of the previous GOP, and (2) reference frames of a given GOP are sent according to their decoding order. However, ensuring the GOP periodicity of the transmitted sequence is more important that satisfying these two guidelines. Thus, these guidelines can be violated if necessary. For example, in Figure 16(b) frames I28, P31, and P34 must be sent in this order according to the first guideline. But the first available slot to send an I frame while maintaining the GOP periodicity is time slot # 42, whereas the server can send a P frame during slot # 39. Thus, the server sends P39 during slot # 39 and I28 during slot # 42, violating the first guideline.

In the process of building up the reference frames at the decoder, the server need not send any B frames in between (otherwise, these B frames are ignored at the decoder). The resulting "empty" slots in the transmission sequence are indicated by 'X's (for 'don't care') in Figure 16(b). Once the required reference frames are received and decoded, BPB can be initiated. Note that some P frames are decoded M periods after they are received, so they must be temporarily stored in their compressed format. After all the reference frames of the previous GOP have been received and decoded, B frames can be received, decoded, and displayed in the backward direction.

Clearly, the management of the frame buffer at the decoder must be modified to support BPB. Instead of storing two reference frames, the decoder must store a maximum of $2N/M$ uncompressed frames. A mechanism is needed to signal to the decoder that the transmitted reference frames are for BPB. Such a mechanism will be described in Section 5. Once the decoder receives an indication that BPB has been requested, it modifies its management of the frame buffer to accommodate up to $2N/M$ uncompressed frames. Figure 17 depicts the change in the content of the frame buffer in our example. Note that the BPB request was issued during the decoding of P40, which under normal playback replaces P34. Thus, by the time the decoder starts modifying its buffer management, the uncompressed P34 (or parts of it) has already been discarded, and P34 must be retransmitted.

4.4.2 Normal Playback to BFS

In addition to reversing the playback direction, a BFS request also involves switching from the normal to the scan version. In this section, we present two different approaches to supporting BFS.

First Approach

In this approach, BFS is initiated at a common I frame. Following a BFS request, normal playback continues until the display of a common I frame. Thus, the server continues sending frames from the normal version for a short period following the receipt of a BFS request. After that, the server switches to the scan version. As in the case of normal-to-BPB, the server must first send all or most of the reference frames of the current *and* previous GOPs of the scan version (for a maximum of $2N/M$ reference frames). Consider the example in Figure 18, in which the scan version has the following parameters: $N = 9$, $M = 3$, and $s = 2$. A BFS request is issued during the playback of frame B72 of the normal version. Normal playback continues until the display of I73; the first common I frame. The server receives the BFS request during the transmission of P76. Thus, all

I1	B2	B3	P4	B5	B6	P7	B8	B9
I10	B11	B12	P13	B14	B15	P16	B17	B18
I19	B20	B21	P22	B23	B24	P25	B26	B27
I28	B29	B30	P31	B32	B33	P34	B35	B36
I37	B38	B39	P40	$\cdots$				

(a) Temporal order of normal version ($N = 9$, $M = 3$).

Received	I37	B35	B36	P40	X	X	P31	X	X	I28	X	X
Decoded	I37	B35	B36	P40	–	–	–	–	–	I28	–	–
Displayed	B33	P34	B35	B36	I37	I37	I37	I37	I37	I37	I37	I37
Time	33	34	35	36	37	38	39	40	41	42	43	44
	(normal playback)			↑ BPB request				(pause)				

Received	P34	X	X	P22	X	X	I19	B36	B35	P25	B33	B32
Decoded	P31	–	–	P34	–	–	I19	B36	B35	P22	B33	B32
Displayed	I37	I37	I37	I37	I37	I37	I37	I37	B36	B35	P34	B33
Time	45	46	47	48	49	50	51	52	53	54	55	56
				(pause)				$\longmapsto$ start BPB				

Received	P13	B30	B29	I10	B27	B26	P16	B24	B23	P4	B21	$\cdots$
Decoded	P25	B30	B29	I10	B27	B26	P13	B24	B23	P16	B21	$\cdots$
Displayed	B32	P31	B30	B29	I28	B27	B26	P25	B24	B23	P22	$\cdots$
Time	57	58	59	60	61	62	63	64	65	66	67	$\cdots$
					(continue BPB)							

(b) Received, decoded, and displayed frames.

Figure 16: Switching from normal playback to backward playback.

Time	Uncompressed frames in buffer
35	I37, P34
36	P40, I37
37	I37 (P40 is discarded)
42	I28, I37
45	P31, I28, I37
48	P34, P31, I28, I37
51	I19, P34, P31, I28, I37
54	P22, I19, P34, P31, I28, I37
57	P25, P22, I19, P34, P31, I28
60	I10, P25, P22, I19, P31, I28
63	P13, I10, P25, P22, I19, I28
66	P16, P13, I10, P25, P22, I19
69	I1, P16, P13, I10, P22, I19

Figure 17: Content of the frame buffer during the transition from normal playback to BPB.

the frames that precede I73 in the playback order have already been sent. The server switches to the scan version and starts sending the reference frames of the current and previous GOPs. In this example, frames I37, I55, P61, and P67 must be received and decoded before BFS is initiated (in general, BFS cannot be initiated before all frames of the current GOP and one or more frames of the previous GOP are decoded). These frames are transmitted following the same guidelines that are used to support BPB. Thus, the transmission order is P61, I55, P67, and I37, while the decoding order is I55, P61, P67, and I37. As in the case of BPB, no B frames need to be transmitted during the buildup of the reference frames. Also, some P frames are decoded few frame periods after they are received, so they must be temporarily stored in their compressed format. During the buildup of the reference frames, the movie pauses at frame 73. Once frame I37 is received and decoded, the transmission, decoding, and reverse playback of the scan version can be streamed, and BFS is initiated. Part (c) of Figure 18 depicts the change in the content of the frame buffer.

In the previous example, the BFS request was issued just before the display of a common I frame, which is a rather best-case scenario. The worst-case scenario occurs when BFS is requested just after the display of a common I frame. In this case, normal playback continues for an additional sN frame periods ($s/2$ seconds) until the next common I frame is encountered. This extra normal playback is followed by a pause period that lasts for no more that two GOPs (one second), during which reference frames are being accumulated in the decoder.

Second Approach

An alternative approach is to initiate BFS from the "closest" reference frame (I or P) of the scan version. When a BFS request is issued, the movie pauses immediately at the currently displayed frame (which could be of any type). The client identifies the reference frame of the scan version that has the closest logical time to (but no larger than) the logical time of the currently displayed frame. BFS is initiated from this reference frame. When the server receives the BFS request, it starts sending the reference frames of the last two GOPs up to (and including) the designated reference frame. Thereafter, the process is similar to the one used in the first BFS approach.

As an example, consider the situation in Figure 19. Since the BFS request is issued during the playback of B80, the movie pauses at that frame. The reference frame of the scan version that is closest (in logical time) to the current logical time is P79. Thus, BFS will be initiated from P79. But before that, the decoder must receive P79, P61, I73, P67, P43, I55, and P49 (in this order); and must decode I73, P79, I55, P61 (in this order). Once this is done, the process of transmitting, decoding, and displaying frames for the purpose of BFS can be streamed, similar to the first approach. The maximum duration of the pause period is given by $2N + M$ frame periods (one second and $M/2N$ of a second), which is independent of s. This is slightly higher than the worst-case pause period in the first approach, but there is no extra normal playback as in the first approach.

I1	B3	B5	P7	B9	B11	P13	B15	B17
I19	B21	B23	P25	B27	B29	P31	B33	B35
I37	B39	B41	P43	B45	B47	P49	B51	B53
I55	B57	B59	P61	B63	B65	P67	B69	B71
I73	B75	B77	P79	B81	B83	P85	B87	B89

(a) Temporal order of the scan version ($N = 9$, $M = 3$, $s = 2$).

Received	···	I73	B71	B72	P76	X	X	P61	X	X	I55	X	X
Decoded	···	I73	B71	B72	P76	–	–	–	–	–	I55	–	–
Displayed	···	B69	P70	B71	B72	I73	I73	I73	I73	I73	I73	I73	I73
Time	···	69	70	71	72	73	74	75	76	77	78	79	80
		(normal playback)			↑				(pause)				
					BFS request								

Received	P67	X	X	P43	X	X	I37	B71	B69	P49	B65	B63	P25
Decoded	P61	–	–	P67	–	–	I37	B71	B69	P43	B65	B63	P49
Displayed	I73	I73	I73	I73	I73	I73	I73	I73	B71	B69	P67	B65	B63
Time	81	82	83	84	85	86	87	88	89	90	91	92	93
				(pause)				⟼					
								start BFS					

Received	B59	B57	I19	B53	B51	P31	B47	B43	···
Decoded	B59	B57	I19	B53	B51	P25	B47	B43	···
Displayed	P61	B59	B57	I55	B53	B51	P49	B47	···
Time	94	95	96	97	98	99	100	101	···
				(continue BFS)					

(b) Received, decoded, and played frames.

Time	Uncompressed frames in buffer
71	I73, P70
72	P76, I73
73	I73 (P76 is discarded)
78	I55, I73
81	P61, I55, I73
84	P67, P61, I55, I73
87	I37, P67, P61, I55, I73
90	P43, I37, P67, P61, I55 (I73 is discarded)
93	P49, P43, I37, P61, I55 (P67 is discarded)
96	I19, P49, P43, I37, I55 (P61 is discarded)

(c) Content of the frame buffer at the decoder.

Figure 18: First approach for switching from normal playback to BFS.

Received	B78	I82	B80	B81	P79	X	X	P61	X	X	I73	X	X
Decoded	B78	I82	B80	B81	–	–	–	–	–	–	I73	–	–
Displayed	B77	B78	P79	B80	B80	B80	B80	B80	B80	B80	B80	B80	B80
	(normal playback)			↑ BFS request					(pause)				

Received	P67	X	X	P43	X	X	I55	X	X	P49	B77	B75	P25
Decoded	P79	–	–	–	–	–	I55	–	–	P61	B77	B75	P67
Displayed	B80	B80	B80	B80	B80	B80	B80	B80	B80	B80	P79	B77	B75
					(pause)						⟼ start BFS		

Received	B71	B69	I37	B65	B63	P31	B59	B57	P7	⋯
Decoded	B71	B69	I37	B65	B63	P43	B59	B57	P49	⋯
Displayed	I73	B71	B69	P67	B65	B63	P61	B59	B57	⋯
					(continue BFS)					

Figure 19: Second approach for switching from normal playback to BFS.

4.5 Switching From BFS to Normal Playback

The easiest way to resume normal playback following BFS is to initiate the normal playback from an I frame that is common to both the normal and the scan versions (this is analogous to the first approach for switching from normal playback to BFS). Thus, when the client requests normal playback, the movie remains in the BFS mode until a common I frame is encountered. At worst, normal playback is resumed at a logical point that is $s/2$ seconds (in movie time) from the logical time at which the resume was requested, but it takes only a maximum of 1/2 second to reach this common I frame (each GOP of the scan version corresponds to a sampled video segment of duration $sN/2N = s/2$ seconds. However, it takes only 1/2 second to play back this GOP). Upon receiving the resume-playback request, the server switches to the normal version, starting from the common I frame. Since two reference frames are needed in the frame buffer to stream the decoding process, the movie pauses at the common I frame for no more that M periods ($M/2N$ of a second). This pause is needed to decode the P frame of the normal version that follows the common I frame. After that, normal playback can be resumed. The switching process is illustrated in the example in Figure 20.

Other forms of interactivity include switching between FFS and BFS without going through normal playback. Also, if the VOD system supports multiple FS speedups, switching can take place between two scan versions that have different speedups. These and other scenarios can be dealt with using similar approaches to the ones we described (with slight modifications to fit the specifics of each scenario). Due to space limitations, we do not elaborate further on these scenarios.

We define the latency of an interactive operation as the difference between its time of request at the client side and its initiation time on the client's display device. This latency measures the actual waiting time of the client. It consists of: (1) roundtrip propagation time (RTT) between the client and the server, (2) processing delay at the server, and (3) "switching delay", which is

Received	I37	B65	B63	P31	B59	B57	P7	B53	B51	I19	B47	B45	X
Decoded	I37	B65	B63	P43	B59	B57	P49	B53	B51	I19	B47	B45	–
Displayed	B69	P67	B65	B63	P61	B59	B57	I55	B53	B51	P49	B47	B45
					(BFS)					↑	(continue BFS)		
								normal playback request					

Received	B41	B39	P40	X	X	X	B38	B39	P43	B41	B42	I46 ···	
Decoded	B41	B39	P40	–	–	–	B38	B39	P43	B41	B42	I46	
Displayed	P43	B41	B39	I37	I37	I37	I37	B38	B39	P40	B41	B42	···
	(continue BFS)				(pause)			$\longmapsto$					
								start normal playback					

Figure 20: Switching from BFS to normal playback.

Requested Operation	Switching Delay (in seconds)
Normal-to-FFS	$s/2$
Normal-to-BPB	$1 + M/2N$
Normal-to-BFS (1st approach)	$1 + s/2$
Normal-to-BFS (2nd approach)	$1 + M/2N$
FFS-to-normal (1st approach)	$1/2$
FFS-to-normal (2nd approach)	$1/2s + M/2N$
BPB-to-normal	$1/2 + M/2N$
BFS-to-normal	$1/2 + M/2N$

Table 2: Worst-case switching delay associated with various interactive operations.

the delay caused by switching from one version to another (it includes the time needed to reach an appropriate switching point and the time needed to build up the frame buffer in BFS operations). The second component of the latency is relatively small, and can be ignored. The RTT depends on the underlying network topology. In [35] the authors report one-way propagation delays from 30 to 50 milliseconds for a wide-area ATM network, and less than 10 milliseconds for ATM LAN connections. Table 2 summarizes the *worst-case* switching delay for various types of requests. This delay is measured in real time (not the logical time of the movie).

For typical values of N, M, and s (say, $N = 15$, $M = 3$ and $s = 4$), the worst-case switching delay associated with common interactive operations ranges from a fraction of a second to three seconds. This delay can be further reduced by using a smaller GOP length (N) or by reducing s. However, reducing the skip factor will increase the storage requirement of the scan version (since more frames are generated), while reducing the GOP length will increase the storage needed for the normal and the scan versions and will potentially reduce the efficiency of the underlying envelope-based scheduling mechanism. Tuning the above parameters requires careful consideration of the involved tradeoffs.

5 Signalling

Signalling between the client and the server must be extended to allow the decoder to distinguish between various versions. For this purpose, we use an in-band signalling mechanism based on the user-data field in the header of an MPEG frame. Each frame carries in its header the value of the skip factor and the playback direction (forward or backward). This information can be conveyed using one byte in the user-data field. The most significant bit of this byte encodes the playback direction while the other seven bits encode the skip factor (in fact, four bits are enough to represent skip factors from $s = 1$ to $s = 16$; the remaining three bits can be used to convey other types of information). For P and B frames of a given version, the skip factor is inserted in the frame header during the encoding of that version. In contrast, for I frames the skip factor is inserted during transmission since some of these frames are common between two or more versions. For all frames, the playback direction is added on the fly during the transmission of the MPEG stream. This can be done efficiently since user data in the frame header are byte aligned and are located at a fixed offset from the beginning of the MPEG frame. The server can insert user-data bytes with minimal parsing of the MPEG stream. Information about the GOP structure of a version is included in the sequence header and in the header of the first GOP. This information can be used to allocate memory for the frame buffer during the initial signalling phase.

6 Striping and Disk Scheduling

An extensive amount of research has been done on the design of multimedia storage systems (see, for example, [17, 18, 23, 24, 34, 39, 12]). In a typical VOD system, each compressed movie is segmented into *blocks*, which are distributed over multiple storage devices (or disks). This process, known as *striping*, is used to increase the disk-retrieval throughput. Striping is being used in many storage systems, such as disk arrays and RAID systems. It is classified into time and space striping. In time striping (e.g., [5, 6]), blocks consist of equal portions of the video movie (e.g., a fixed number of successive frames), so that the number of bytes per block varies from one block to another. In space striping (e.g., [16, 28]), blocks contain the same number of bytes, but this number may correspond to different numbers of frames per block. Time striping has the advantage that the retrieved blocks can be easily streamed into the network at a fixed frame rate. To maintain a constant disk access time, disk scheduling policies are needed. In time striping, video frames within a block are stored contiguously to optimize the access time of the disk. For MPEG video, a block usually consists of a number of consecutive GOPs.

In [8], Chen proposed an attractive disk scheduling scheme for MPEG video, in which MPEG frames within a block are organized in the order of their significance, with B frames placed at the end of the block. This scheme has the advantage that when disk resources are oversubscribed, B

frames can be dropped to allow for the retrieval of the more important I and P frames. We extend this scheme to accommodate the scan versions of a movie. Let $s_1, s_2, \ldots, s_k$ be the k skip factors supported by the system, with $1 < s_1 < s_2 < \cdots < s_k$. Let s_{lcm} be the least common multiple of these skip factors. Each block consists of s_{lcm} GOPs from the normal version plus s_{lcm}/s_i GOPs from the ith scan version, for all $i = 1, \ldots, k$. This way, each block contains portions of the normal and scan versions that correspond to the same segment of the movie. The resulting block consists of $(s_{lcm} + \sum_i s_{lcm}/s_i)N$ video frames. Increasing the size of the retrieval block has the disadvantages of reducing the interleaving granularity of the disk subsystem as well as increasing the amount of memory needed to temporarily store the retrieved blocks at the server before being sent over the network. Nonetheless, a reasonable block size can be achieved with proper selection of the skip factors. Frames within a block are organized as follows. First, the I frames of the s_{lcm} GOPs of the normal version are put at the beginning of the block (no separate I frames are generated for the scan versions since such frames are already included in the normal version). They are followed by P frames; first the ones that belong to the normal version, then the ones that belong to the scan versions. Finally, B frames of the normal and scan versions are added, in the indicated order. This structure allows for efficient disk access since relevant data are stored consecutively on disk and no extra disk-head movements are needed to access "out-of-stream" I frames. One can also adapt this structure to multiple disk systems, which also support the notion of a retrieval block.

In the previous striping scheme, only a fraction of the frames in a block are sent over the network. This results in disk-bandwidth loss, which is comparable to the storage overhead of the scan versions (about 30%-50% for reasonable parameter values). To improve the utilization of the disk bandwidth, a different striping strategy can be used. In this strategy, all I frames are placed at the beginning of the block (as before). However, these I frames are followed by P and B frames of each version, in the following order: first, P and B frames of the normal version (in this order), followed by P and B frames of the first scan version, followed by P and B frames of the second scan version, and so on until the last scan version. The advantage of this organization is that it allows the server to interrupt the transfer of the block as soon as the desired frames have been read into memory. For example, during normal playback, only I, P, and B frames of the normal version are retrieved. During a FS operation with speedup s_1, a larger part of the block is read from disk. In addition to I, P, and B frames of the normal version, this part includes P and B frames of the first scan version. There is still some loss in disk bandwidth. But this loss occurs only during fast-scanning periods, which amount to a small fraction of the playback time.

7 Admission Control

As described in Section 2, the VOD distribution network consists of a remote server, several head-end (HE) switches, and bandwidth pipes that extend from the server to these HE switches. A VOD client requests a movie through one of the HE switches. The switch propagates the request back to the server, which checks its admissibility. If the pipe onto which the prospective stream is to be multiplexed has sufficient capacity, then the request is accepted. Note that the admission test is done online, so it must be performed fairly quickly. Once a request is accepted, the new stream is scheduled for multiplexing onto one of the bandwidth pipes (as explained in Section 2).

In the absence of interactivity, the admission control test is rather simple. Let C_{tot} be the total capacity of the examined bandwidth pipe onto which n streams are being multiplexed. The used bandwidth is given by $nC(\boldsymbol{u}, n)$, where $C(\boldsymbol{u}, n)$ is the per-stream allocated bandwidth (PSAB), defined in (1). To minimize $C(\boldsymbol{u}, n)$ the server implements one of several stream-scheduling schemes [40]. In effect, a stream-scheduling scheme aims at optimizing the vector $\boldsymbol{u}$ such that $C(\boldsymbol{u}, n)$ is minimized for a fixed n. The scheduling scheme is often suboptimal in the sense that the resulting $C(\boldsymbol{u}, n)$ is minimal only for large n. However, even for small values of n, the resulting $C(\boldsymbol{u}, n)$ is close to its minimum value. When traffic streams are modeled by global envelopes, the minimum $C(\boldsymbol{u}, n)$ is fixed for a given n. On the other hand, when the streams are modeled by window-based envelopes, the minimum $C(\boldsymbol{u}, n)$ varies from one window to another. In either case, the idle capacity of the bandwidth pipe is given by $C_{tot} - nC(\boldsymbol{u}, n)$. The value of $\boldsymbol{u}$ (and consequently $C(\boldsymbol{u}, n)$) depends on the particular scheduling scheme. For example, under *minimum-rate phase* (MRP) scheme [40], a new stream is scheduled to start in the frame period $i \in \{1, 2, \ldots, N\}$ of the next GOP for which $\overline{b}_{tot}(t)$ is minimal over all frame periods of that GOP (see (1) for definition of $\overline{b}_{tot}(t)$). Other scheduling schemes differ in the choice of the GOP interval(s) over which the minimization is performed. In any case, the new $(n+1)$th stream is admitted if and only if $C_{tot} - (n+1)C(\boldsymbol{u}, n+1) \geq 0$ for all subsequent time windows.

With the inclusion of interactivity, the situation is slightly different. In contrast to a playback-only VOD service, the traffic pattern in interactive VOD cannot be determined in advance since it depends on the user behavior. Unless some restrictions are imposed (e.g., a FFS operation is limited to x minutes), one must account for all possible patterns of user's behavior. Let's first define a video *clip* as a set of frames that are transported during one time window at a constant frame rate. Under window-based envelopes, the parameters of the envelope varies from one clip to another. In the following, both the normal and scan versions are characterized by window-based envelopes (global envelopes are special cases of window-based envelopes).

Suppose that all forms of interactivity are supported by the system (including forward and backward 'fast jump'). This means that during a given time window, the client could be viewing any clip of any version of the movie. Accordingly, the worst-case envelope of the transmitted stream is obtained by taking maximum values of the envelopes of all clips and of all versions. If the envelopes of the scan versions are bounded by the envelope of the normal version (i.e., $S_B^{(u)} = S_P^{(u)} = 0$), then the worst-case envelope is simply the global traffic envelope of the normal version, and the admission test is similar to the one used for a playback-only VOD service under global envelopes.

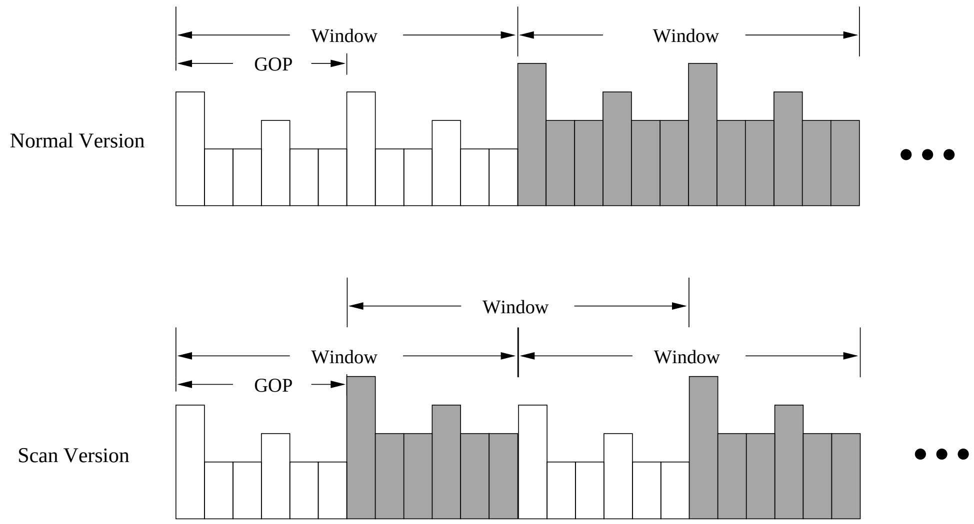


Figure 21: Computation of the worst-case traffic envelope based on the envelopes of the normal version and a scan version with $s = 2$.

If $S_P^{(u)}$ or $S_B^{(u)}$ have nonzero values, then the window-based envelopes of the *scan* versions will have some impact on the shape of the worst-case envelope. To quantify this impact, we consider the situation when switching between versions occur at common I frames. Potential switching points are Ns_{lcm} frames apart, where s_{lcm} is the least common multiple of all supported skip factors. Without loss of generality, we set the size of a clip (and thus, the window interval) to a multiple of Ns_{lcm} frames. Since the clip size is the same for both the normal and scan versions, a clip in the ith scan version with skip factor s_i corresponds to the same video portion represented by s_i clips of the normal version. To accommodate switching between normal and scan versions at appropriate I frames, we allow the clips of a scan version to overlap, as shown in Figure 21 for $s = 2$. This results in normal and scan versions with equal number of clips (although the normal version has more frames). The worst-case envelope is then obtained by overlapping all the clips of all windows and of all versions, resulting in a worst-case *global* traffic envelope that can be used in the admission test.

8 Summary and Future Work

In this paper, we presented an approach for supporting interactive fast-scanning (FS) operations in a VOD system. This approach is integrated into a previously proposed framework for distributing archived, MPEG-coded video streams over a wide-area network. Scanning operations are supported by generating multiple, differently encoded versions of each movie. In addition to a *normal* version that is used for normal playback, several *scan versions* are maintained at the server. Each scan version is obtained by encoding a sample of the raw frames, and is used to support both forward and backward fast scanning at a given speedup. The server responds to a FS-related request by switching from the currently transmitted version to another version. By proper encoding of the scan versions, interactive scan operations can be supported with little or no extra bandwidth and with the same decoding requirement of normal playback. This gain comes at the expense of small storage overhead at the server and some variability in the quality of the motion picture during the fast-scanning periods. Our scheme does not impose any restriction on the number, spacing, or sequencing of interactive operations.

VOD clients should be given the flexibility to choose from a set of available VOD services that offer different levels of interactivity. Billing would then be done based on the quality and flexibility associated with the selected service. Our future work includes developing a multi-level QoS framework for interactive VOD. Each level corresponds to a certain degree of interactivity, which could include some limitations on the interactive functions (e.g., number of supported speedups, visual quality during scanning, maximum duration of a scanning operation, etc.).

References

[1] C. Aggarwal, J. Wolf, and P. Yu. On optimal batching policies for video-on-demand servers. In *IEEE Multimedia Computing and Systems Conference*, pages 253–258, 1996.

[2] K. C. Almeroth and M. H. Ammar. The use of multicast delivery to provide a scalable and interactive video-on-demand service. *IEEE Journal on Selected Areas in Communications*, 14(6):1110–1122, Aug. 1996.

[3] D. Anderson. Metascheduling for continuous media. *ACM Transactions on Computer Systems*, 11(3):226–252, 1993.

[4] ATM Forum. Audiovisual multimedia services: Video on demand specification 1.1, Mar. 1997.

[5] C. Bernhardt and E. Biersack. *The server array: A scalable video server architecture.* Kluwer Press, 1996.

[6] M. M. Buddhikot and G. M. Parulkar. Efficient data layout, scheduling, and playout control in MARS. In *Proceedings of NOSSDAV '95 Conference*, 1995.

[7] Y. H. Chang et al. An open-systems approach to video on demand with VCR like functions. *IEEE Communications Magazine*, 32(5):68–80, May 1994.

[8] H.-J. Chen. *A Disk Scheduling Scheme and MPEG Data Layout Policy for Interactive Video Access from a Single Disk Storage Device.* PhD thesis, Boston University, College of Engineering, 1995.

[9] H. J. Chen, A. Krishnamurthy, T. Little, and D. Venkatesh. A scalable video-on-demand service for the provision of VCR-like functions. In *Proceedings of the IEEE Multimedia Conference*, 1995.

[10] M.-S. Chen, D. D. Kandlur, and P. S. Yu. Support for fully interactive playout in a disk-array-based video server. In *Proceedings of the Second International Conference on Multimedia*, pages 391–398, Oct. 1994.

[11] M.-S. Chen, D. D. Kandlur, and P. S. Yu. Storage and retrieval methods to support fully interactive playout in a disk-array based video server. *Multimedia Systems Journal*, 3:126–135, 1995.

[12] C.-Y. Cheng, Y.-J. Oyang, and M.-H. Lee. Disk system design for guaranteeing quality of service in interactive video-on-demand systems. In *Proceedings of the IS&T/SPIE Storage and Retrieval for Image and Video Databases IV Conference*, pages 200–207, 1998.

[13] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *ACM Multimedia Conference*, pages 15–24, 1994.

[14] J. K. Dey-Sircar, J. D. Salehi, J. F. Kurose, and D. Towsley. Providing VCR capabilities in large-scale video servers. In *Proc. ACM Multimedia '94*, pages 25–32, 1994.

[15] W.-C. Feng and J. Rexford. A comparison of bandwidth smoothing techniques for the transmission of prerecorded compressed video. In *Proceedings of INFOCOM '97*, Apr. 1997.

[16] C. S. Freedman and D. J. DeWitt. The SPIFFI scalable video-on-demand system. In *Proceedings of ACM SIGMOD '95 Conference*, June 1995.

[17] D. J. Gemmell, H. M. Vin, and P. V. Rangan. Multimedia storage servers: A tutorial. *Computer Magazine*, pages 40–49, May 1995.

[18] L. Golubchik, J. Lui, and R. Muntz. Reducing I/O demand in video-on-demand storage servers. In *Proccedings of the ACM Sigmetrics/Performance '95 Conference*, pages 25–36, 1995.

[19] L. Golubchik, J. Lui, and R. Muntz. Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers. *Multimedia Systems Journal*, 4(3):140–155, 1996.

[20] M. Graf. VBR video over ATM: Reducing network resource requirements through endsystem traffic shaping. In *Proc. of IEEE INFOCOM '97*, pages 48–57, 1997.

[21] ISO/MPEG II. ISO CD11172-2: Coding of moving pictures and associated audio, Dec. 1992.

[22] J. R. Jones. Baseband and passband transport systems for interactive video services. *IEEE Communications Magazine*, pages 90–101, May 1994.

[23] D. Kandlur, M. Chen, and Z. Shae. Design of multimedia storage server. In *Proceedings of IS&T/SPIE Symposium on Elec. Imaging Sci. and Tech*, Feb. 1994.

[24] K. Keeton and R. H. Katz. Evaluation of video layout strategies for a high-performance storage server. *ACM Multimedia Systems Journal*, 3:43–52, May 1995.

[25] M. Krunz and S. K. Tripathi. Exploiting the temporal structure of MPEG video for the reduction of bandwidth requirements. In *Proc. of IEEE INFOCOM'97*, pages 67–74, Kobe, Japan, Apr. 1997.

[26] M. Krunz and S. K. Tripathi. Impact of video scheduling on bandwidth allocation for multiplexed MPEG streams. *ACM Multimedia Systems Journal*, 5(6):347–357, Dec. 1997.

[27] S. S. Lam, S. Chow, and D. K. Y. Yau. An algorithm for lossless smoothing for MPEG video. In *Proceedings of the ACM SIGCOMM '94 Conference*, Aug. 1994.

[28] Y. B. Lee and P. C. Wong. A server array approach for video-on-demand service on local area networks. In *Proceedings of IEEE INFOCOM '96 Conference*, Mar. 1996.

[29] V. Li, W. Liao, X. Qiu, and E. Wong. Performance model of interactive video-on-demand systems. *IEEE Journal on Selected Areas in Communications*, 14(6):1099–1109, Aug. 1996.

[30] W. Liao and V. Li. The split and merge (SAM) protocol for interactive video-on-demand systems. In *Proceedings of INFOCOM'97*, Apr. 1997.

[31] S. C. Liew and C. yin Tse. Video aggregation: Adapting video traffic for transport over broadband networks by integrating data compression and statistical multiplexing. *IEEE Journal on Selected Areas in Communications*, 14(6):1123–1137, Aug. 1996.

[32] J. M. McManus and K. W. Ross. Video-on-demand over ATM: Constant-rate transmission and transport. *IEEE Journal on Selected Areas in Communications*, 14(6):1087–1098, Aug. 1996.

[33] B. Ozden, A. Biliris, and R. R. A. Silberschatz. A low-cost storage server for movie on demand databases. In *Proc. of the 20th VLDB Conference*, 1994.

[34] P. V. Rangan and H. M. Vin. Efficient storage techniques for digital continuous multimedia. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):564–573, Aug. 1993.

[35] D. J. Reininger, D. Raychaudhuri, and J. Y. Hui. Bandwidth renegotiation for VBR video over ATM networks. *IEEE Journal on Selected Areas in Communications*, 14(6):1076–1086, Aug. 1996.

[36] J. D. Salehi, Z.-L. Zhang, J. F. Kurose, and D. Towsley. Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing. In *Proc. of the ACM SIGMETRICS/PERFORMANCE '96 Conference*, May 1996.

[37] P. J. Shenoy and H. M. Vin. Efficient support for scan operations in video servers. In *Electronic Proceedings of ACM Multimedia '95*, Nov. 1995.

[38] M. Vernick, C. Venkatramani, and T. cker Chiueh. Adventures in building the Stony Brook video server. In *Proceedings of ACM Multimedia*, Nov. 1996.

[39] H. M. Vin and P. V. Rangan. Designing a multiuser HDTV storage server. *IEEE Journal on Selected Areas in Communications*, 11(1):153–164, 1993.

[40] Z. Wei, M. Krunz, and S. Tripathi. Efficient transport of stored video using stream scheduling and window-based traffic envelopes. In *Proceedings of the IEEE ICC '97*, 1997.

[41] P. S. Yu, J. L. Wolf, and H. Shachanai. Design and analysis of a look-ahead scheduling scheme to support pause-resume for video-on-demand application. *Multimedia Systems Journal*, 3(4):137–150, 1995.