Dynamic Programming-Based Reverse Frame Selection for VBR Video Delivery Under Constrained Resources

Dayong Tao, Jianfei Cai, *Member, IEEE*, Haoran Yi, Deepu Rajan, Liang-Tien Chia, and King Ngi Ngan, *Fellow, IEEE*

***Abstract*—In this paper, we investigate optimal frame-selection algorithms based on dynamic programming for delivering stored variable bit rate (VBR) video under both bandwidth and buffer size constraints. Our objective is to find a feasible set of frames that can maximize the video's accumulated motion values without violating any constraint. It is well known that dynamic programming has high complexity. In this research, we propose to eliminate nonoptimal intermediate frame states, which can effectively reduce the complexity of dynamic programming. Moreover, we propose a reverse frame selection (RFS) algorithm, where the selection starts from the last frame and ends at the first frame. Compared with the conventional dynamic programming-based forward frame selection, the RFS is able to find all of the optimal results for different preloads in one round. We further extend the RFS scheme to solve the problem of frame selection for VBR channels. In particular, we first perform the RFS algorithm offline, and the complexity is modest and scalable with the aids of frame stuffing and nonoptimal state elimination. During online streaming, we only need to retrieve the optimal frame-selection path from the pregenerated offline results, and it can be applied to any VBR channels as long as the VBR channels can be modeled as piecewise CBR channels. Experimental results show good performance of our proposed algorithms.**

***Index Terms*—Bandwidth smoothing, dynamic programming, motion awareness, optimal frame selection, variable bit rate (VBR) channels, VBR video delivery.**

I. Introduction

A VARIABLE BIT RATE (VBR) encoded video generally offers improved picture qualities over the corresponding constant-bit-rate (CBR) encoded video give the same average bit rate [1], [2]. However, the VBR video traffic is more difficult to manage because of its significant bit-rate burstiness over multiple time scales [3]–[5]. In particular, the high peak and bursty bit rate can substantially increase the bandwidth requirement for the continuous playback at the client site. To address this problem, various bandwidth smoothing techniques [6]–[10] have been proposed. The basic idea of bandwidth smoothing is to prefetch data ahead of each burst so that large frames can be transmitted earlier at a slower rate. Most existing smoothing techniques focus on either minimizing the bandwidth requirements at a given buffer size [11], [12] or minimizing the buffer requirements under rate-constrained bandwidth conditions [13]. While bandwidth limits the amount of data that can be transmitted per unit time, buffer size regulates the amount of data that can be prefetched [14]. If both bandwidth and buffer size are limited, lossy smoothing is unavoidable.

Given the maximum bandwidth and fixed buffer size, Ng and Song [15] suggested to introduce playback pauses or delete B-frames (and subsequently P-frames) when the transmission exceeds the rate limit. Their algorithms drop frames without content awareness and have no global optimization criteria. In [14], Zhang *et al.* proposed an optimal selective frame discard algorithm to minimize the number of frames that must be discarded in order to meet the bandwidth and buffer size limits. However, their algorithm does not take into account semantic frame importance and only considers motion JPEG videos. In [16], Zhou and Liou proposed a nonlinear frame sampling strategy for video streaming under bandwidth and buffer constraints. Their objective is to obtain an optimal set of frames that can maximize the video's salient scores. Dynamic programming is used to find the optimal path. Nevertheless, the authors did not consider the inter-frame dependency, and they focused on videos with constant frame sizes.

In addition to the individual problems pointed out above, most of the existing *lossy smoothing* algorithms assume CBR channels during the smoothing process. However, in reality, the network bandwidth such as Internet bandwidth is usually time-varying. In [17], Feng and Liu proposed two methods for streaming stored video over VBR channels (both methods precompute a bandwidth smoothing plan assuming fixed buffer size and constant bandwidth): 1) adapt the video stream on the fly and 2) run the smoothing algorithm online under the new bandwidth condition for the rest of the frames. However, real-time computation of the transmission plan is too complicated for timely delivery, and the situation becomes even worse when there are many concurrent client connections. In [18], Gan *et al.* proposed a more robust dual-plan bandwidth-smoothing method for layer-encoded video streaming. Upon bandwidth renegotiation failure, the scheme adaptively discards the enhancement-layer data to maintain the original frame rate.

Another problem of most existing lossy smoothing algorithms is that they usually do not consider the packet loss problem caused by network congestion or physical-layer bit corruptions. Recently, we have seen extensive studies on

Manuscript received January 7, 2006; revised July 1, 2006. This work was supported in part by Singapore A*STAR SERC under Grant 032 101 0006. This paper was recommended by Associate Editor D. O. Wu.

D. Tao, J. Cai, H. Yi, D. Rajan, and L.-T. Chia are with the School of Computer Engineering, Nanyang Technological University, 63798 Singapore (e-mail: asjfcai@ntu.edu.sg).

K. N. Ngan is with The Chinese University of Hong Kong, Hong Kong.

Color versions of Figs. 3, 5, 7, and 10–15 are available at http://ieeexplore.org.

Digital Object Identifier 10.1109/TCSVT.2006.884568

rate-distortion optimized (RDO) video streaming over lossy channels [19]–[21]. The most representative work is the one in [19], where Chou *et al.* proposed a framework for streaming packetized media over a lossy packet network in an RDO way. The proposed framework is able to minimize the end-to-end distortion under a rate constraint by choosing the right packets to transmit at a given transmission opportunity. Although the framework is very comprehensive and theoretically sound, it requires an accurate network delay model, which is very difficult to obtain for a network such as the Internet. In addition, the proposed optimal packet scheduling in [19] is very complex, which might limit its implementation in practice.

In this paper, we assume that the packet loss problem can be well handled by the error control techniques deployed in the transportation layer and the link layer. We only focus on optimal lossy smoothing based on the *a priori* motion information in video. By lossy smoothing, we mean that not all of the frames can be selected due to resource constraints. Since high motion objects are usually more perceptible to human eyes, it is desired to select more frames in the high motion segments for better perception. Our goal is to select a set of frames out of the video that can deliver the maximal accumulated motion metrics while being guaranteed transmittable and playable under bandwidth and buffer constraints.

In particular, we first analyze the problem of delivering stored video over CBR channels. In addition to the frame stuffing approach [16], we propose to eliminate nonoptimal intermediate frame states in forward frame selection to reduce the computation complexity of dynamic programming. Then, we find that the problem can also be solved by a reverse frame selection (RFS) scheme, where the selection starts from the last frame and ends at the first frame. The major advantage of our proposed RFS is that, by running RFS just once, we can easily retrieve any optimal frame-selection path starting from any frame at any buffer state. We further extend the RFS scheme to solve the problem of streaming stored video over the VBR channels that can be regarded as piecewise CBR channels. We only need to run RFS k times, where k is the number of channel bandwidth samples, and it can apply to any pattern of the VBR channels with bandwidth changes occurring at any time.

The remainder of this paper is organized as follows. Section II states the problem setting and introduces the related work. Section III reviews our previous work for computing the amount of motion in each frame. We describe our proposed forward frame selection and RFS algorithms for CBR channels in Sections IV and V. Section VI describes how to apply the RFS scheme for the VBR channels. In Section VII, we evaluate the performance of our proposed algorithms under both CBR and VBR channels. Finally, Section VIII concludes this paper.

II. Background

A. Problem Statement

Our optimal transmission plan is computed based on the system setting shown in Fig. 1. Two separate buffers are used at the client side for smoothing and decoding purposes, respectively. The decoding buffer retrieves compressed frames from the receiving buffer and sends the decoded pictures to video

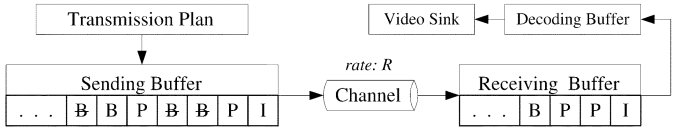


Fig. 1. System setting for computing the optimal frame-selection path.

sink for display. We assume that, once a frame is retrieved from the receiving buffer, its space is immediately made available for future incoming data. In other words, we only need to examine the receiving buffer fullness to avoid buffer overflow and underflow when we compute the transmission plan. In the following, without specification, buffer size means the receiving buffer size.

In addition, for practical video coding, there usually exists inter-frame dependencies in the coded video. For example, most MPEG videos consist of I-, P-, and B-frames. While I-frames are intra-coded and can be decoded independently, forward predicted P- and bidirectionally predicted B-frames need their references for proper decoding. Thus, the encoding order is different from the display order. In this research, we select frames according to the encoding order. In other words, for the receiving buffer, frames are removed one by one in their encoding order at fixed intervals. It is the decoding buffer's responsibility to hold necessary references.

The transmission plan consists of the optimal frame-selection path and the schedule for frame delivery. The schedule tells the server the time and the period to stop transmitting data. In particular, for a long sequence of small-size frames being transmitted, the client consumes less data than the amount of data being received, which might cause buffer overflow eventually. Under such a circumstance, the server will have to either stay idle for some time or transmit at a reduced rate to prevent client buffer overflow.

After describing the system setup, now we formulate the problem. For a video sequence with N frames, let B denote the size of the client buffer and f_i denote the frame size for the ith frame, where $i = 1, 2, \ldots, N$. The problem of motion-based optimal frame selection can be expressed as

$$\max_{s_i} M = \sum_{i=1}^{N} m_i \cdot s_i \tag{1}$$

subject to the bandwidth constraint

$$r_i \leq \text{bandwidth/framerate} \tag{2}$$

and the buffer constraint for $k = 1, 2, \ldots, N$

$$\sum_{i=1}^{k} f_i \cdot s_i - \text{preload} \leq \sum_{i=1}^{k} r_i \leq B - \text{preload} + \sum_{i=1}^{k-1} f_i \cdot s_i \tag{3}$$

where m_i is the motion metric gain for selecting the ith frame, s_i is the indicator function, which is equal to 1 if the ith frame is selected and equal to 0 otherwise, and r_i is the amount of data

sent within the time slot between the $(i-1)$th frame and ith frame. Note that in this paper we consider both CBR and VBR channels. In the cases of VBR channels, *bandwidth* is a variable. However, we assume that the bandwidth does not change rapidly and remains constant in a magnitude of a few seconds. This is reasonable for slow-fading channels in mobile networks or using TFRC [22] in the Internet, where the application rate is adjusted according to a certain feedback interval. In other words, the VBR channels we consider in this paper can be regarded as piecewise CBR channels so that dynamic programming can be applied.

B. Related Work on Frame Selection

Here, we describe three existing frame-selection algorithms: the just-in-time (JIT) algorithm [14], the greedy algorithm [14], and the Z-B diagram algorithm with frame stuffing [16], which will be used for comparison with our proposed algorithms. Since the original algorithms only consider intra-frame video coding, we make some changes in order for them to be applicable with inter-frame dependencies.

1) JIT Algorithm: The JIT algorithm [14] is probably the most intuitive approach for frame selection. It always drops the current frame if client buffer underflow occurs and reduces the transmission rate when buffer overflow occurs. Consequently, the JIT algorithm has no content awareness. Due to inter-frame dependency, we orderly apply the JIT algorithm to different types of frames, I, P and B, and make sure the references are selected before we select a new frame. The computational complexity of the layered JIT algorithm is $O(N^2)$.

2) Greedy Algorithm: The greedy algorithm proposed in [14] is always trying to make the result look the best at the moment. It selects frames according to their reward metrics, i.e., the one currently having the largest reward metric will get selected first. To overcome inter-frame dependency, we use weighted metric for frame sorting, which is given by

$$m_i^w = \begin{cases} m_i, & \text{if frame } i \text{ is B frame} \\ \sum_{k=i}^{j} m_k, & \text{if frame } i \text{ is I- or P-frame} \end{cases} \quad (4)$$

where j is the index for the last frame in the current GOP. In this way, we ensure that the reference frames are always given greater weighted metric and thus are considered first. However, we still need to check whether a frame's references have been selected or not before inserting the frame to the path. The overall computational complexity of the greedy algorithm is also $O(N^2)$. In addition, in this paper we introduce another metric $m_i^r = m_i/f_i$ instead of using the original motion metric m_i for frame selection, since the greedy algorithm is very sensitive to frame size.

3) Z-B Diagram: Fig. 2 shows the Z-B diagram proposed in [16]. In particular, a discrete-time model is used at frame level for client buffer management. Each discrete-time point along the horizontal direction is identified by the particular frame fetched out at that moment for decoding, and each buffer fullness level at any time point is called a state indicated by an arrow endpoint in Fig. 2. As shown in the figure, all of the enqueue lines

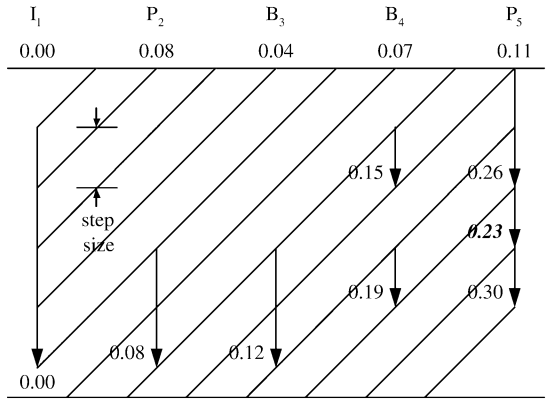


Fig. 2. Z-B diagram with frame stuffing.

(slanted lines) are vertically separated by a fixed distance called the step size and every state at each frame is on an enqueue line. This configuration effectively bounds the number of states at each frame by $B/\text{stepsize}$. The larger the step size, the fewer the number of states and, hence, the less the amount of computational work. To realize the configuration, every frame's size must be a multiple of the step size. In the case of VBR video, the authors [16] suggested to use the greatest common divisor (GCD) of all the frame sizes as the step size. However, in practice, the GCD is most likely to be a very small value, which results in a large number of states at each frame time point and thus significantly increases the computational complexity. Hence, frame stuffing has to be used to increase the step size at the cost of sacrificing bandwidth. For example, suppose stepsize $= 100$, a frame size of 1009 will be stuffed with 91 dummy data to make its size 1100. The average stuffing for a video of N frames is $N \times ((\text{stepsize} - 1))/(2)$.

III. Motion Information Representation

As stated in Section II-A, a fundamental problem we need to solve is how to represent the amount of motion for each video frame. The common approach is to analyze the motion field and use the motion energy to quantify the amount of motion such as in [23]. In this paper, we apply our previous work, "Pixel Change Map" (PCM) [24], [25], to compute the amount of motion. Compared with the approaches directly based on motion fields, the PCM scheme is of low complexity and very easy to implement. In fact, any frame or content classification scheme can be used in our proposed frame-selection algorithms. The PCM scheme by no means is the only or the best way to measure the content importance.

A. Pixel Change Map

The perception of motion content for the human visual system relies on the intensity of the motion. By intensity, we mean how fast a certain object moves. The faster the object moves, the more perceptible it is to humans. As we have observed that a higher intensity of motion would lead to a large number of pixel changes in the video frames, the pixel change map of the frame gives a good characterization of the motion content in the video.

This can be also justified from the famous optical flow constraint equation [26]

$$-d = [p_x, p_y] \cdot [v_x, v_y]^T \tag{5}$$

which shows that the velocity of a pixel $[v_x, v_y]$ in a video signal $[p_x, p_y]$ is directly related to the temporal derivative d, which is approximated by the difference of adjacent frames.

B. Motion Curve

Based on the above observation, we use the simple PCMs as the measurement of the amount of motion in video signals. In particular, for the current frame i, we compute the frame absolute differences $d_i(x, y)$, where $d_i(x, y) = |p_i(x, y) - p_{i-1}(x, y)|$. For each pixel in this frame, if the absolute difference d_i is greater than a fixed threshold of 10, the corresponding location in the PCM is set to 1. The comparison of d_i with the threshold is simply to undo the effect of any noise associated with the camera or the discretization process when dealing with digital camera. The reason we choose the threshold of 10 is that this threshold has been found to be quite robust to noise in our earlier work [25].

After getting the PCMs, we form a 1-D pixel change sequence, where the ith component denoted as $\overline{d}_i$ is the average pixel change in the ith PCM. Then, we filter this pixel change sequence to obtain a more accurate measurement of motion since the human's perception of motion content in the video has the "smoothening" effect, and the human eyes tend to smooth the motion of the video [27]. Besides, the PCMs also contain pixel changes due to other factors in addition to motion such as sudden change of lighting conditions. Those pixel changes are regarded as noise that corrupts the true measurement of the amount of motion. To get the accurate measurement of the video motion content according to human perception, we use filters to remove the noise from the pixel change maps. Details on the filter design can be found in [24] and [25].

In this paper, we simply define the motion metric gain m_i in (1) as $m_i = \tilde{d}_i$, where $\tilde{d}_i$ is the filtered average pixel change values. Using the proposed PCM scheme, we extract the "motion curves" for four common interchange format (CIF) video sequences: 1) Akiyo, 2) Foreman, 3) Mobile, and 4) Stefan. Fig. 3 shows the motion curves for each video separately. We evaluate the effectiveness and accuracy of the "motion curves" by watching the video evolving with the "motion curves." We find that the extracted motion curves correspond to the motion content in the videos very well. In particular, for the Akiyo video sequence, the video contains very little motion and thus the motion curve [Fig. 3(a)] is very close to zero. The "Foreman" video sequence contains various amount of motion at different time, especially when there is a large camera panning motion from frame 175 to frame 235, which is represented as a plateau in the motion curve [Fig. 3(b)]. For the Mobile video sequence, which contains very smooth object motion and camera motion, the extracted motion curve [Fig. 3(c)] is relatively flat. For the Stefan video sequence, the motion content in the video is very high and there is a periodical motion due the rhythm of playing tennis. As shown in Fig. 3(d), we can see that the extracted motion curve manifests this periodical rhythm very well. At the end of the video sequence, the player rushes towards the net and we see a up drift of the motion curve there, which indicates the increasing amount of motion.

IV. Forward Frame Selection for CBR Channels

After obtaining the motion metrics, here we discuss how to maximize the reward function shown in (1) by selecting a feasible set of frames, which satisfies the fixed network bandwidth and the buffer constraints. Since each video frame is either selected or discarded, this problem can be considered as a 0–1 *knapsack problem* [28], which can be solved by dynamic programming. The Viterbi algorithm is a dynamic programming algorithm often used for solving optimization problems whose solutions depend on their subproblems [28]. It avoids overlap computation by solving each subproblem once and saves the answer to a table for future usage. At the final stage, it performs back tracing to find the optimal path that reaches the optimal solution.

Without using the Viterbi algorithm, theoretically the number of possible path is 2^N, which means the computational complexity grows exponentially with the number of frames. By using the Viterbi algorithm, the complexity can be greatly reduced to *O(BN)*, where *B* represents the client buffer size. In this section, we introduce our proposed dynamic programming-based forward frame-selection algorithm, which consists of three basic components: nonoptimal state elimination, virtual states, and optimal preload. The component of nonoptimal state elimination is for reducing the complexity of dynamic programming. The complexity can be further decreased by combining with the frame stuffing approach mentioned in [16]. The component of virtual states is to deal with the issue of inter-frame dependency, and the component of optimal preload is to find the lowest preload value for the global optimal result.

A. Viterbi Trellis

Similar to the Z-B diagram algorithm [16], we also use the common discrete-time model at frame level for client buffer management, as shown in Fig. 4. Let b_i^j denote the buffer fullness level at the jth state at frame i. If state l at frame q is created by state k at frame p, or in other words state l at frame q is directly lined with state k at a previous frame p, then we have the following relation:

$$b_q^l = \min\left\{b_p^k + (q-p)(\text{bandwidth/framerate}) - f_q, B - f_q\right\} \tag{6}$$

where bandwidth/framerate is the amount of data that can be transmitted in one frame time-slot period, B is buffer size, and f_q is the size of frame q. Note that b_q^l becomes a full buffer state ($b_q^l = B - f_q$) if buffer overflow occurs during state transition from b_p^k to b_q^l. In this case, the server has to stay idle for some time or transmit at a reduced rate in order to avoid client buffer overflow. In addition, there is a preload level at the initial stage just before playback starts. It is the amount of data that has been prefetched into the client buffer. The time required to buffer preload is called startup delay

$$\text{startup delay} = \text{preload/bandwidth}. \tag{7}$$

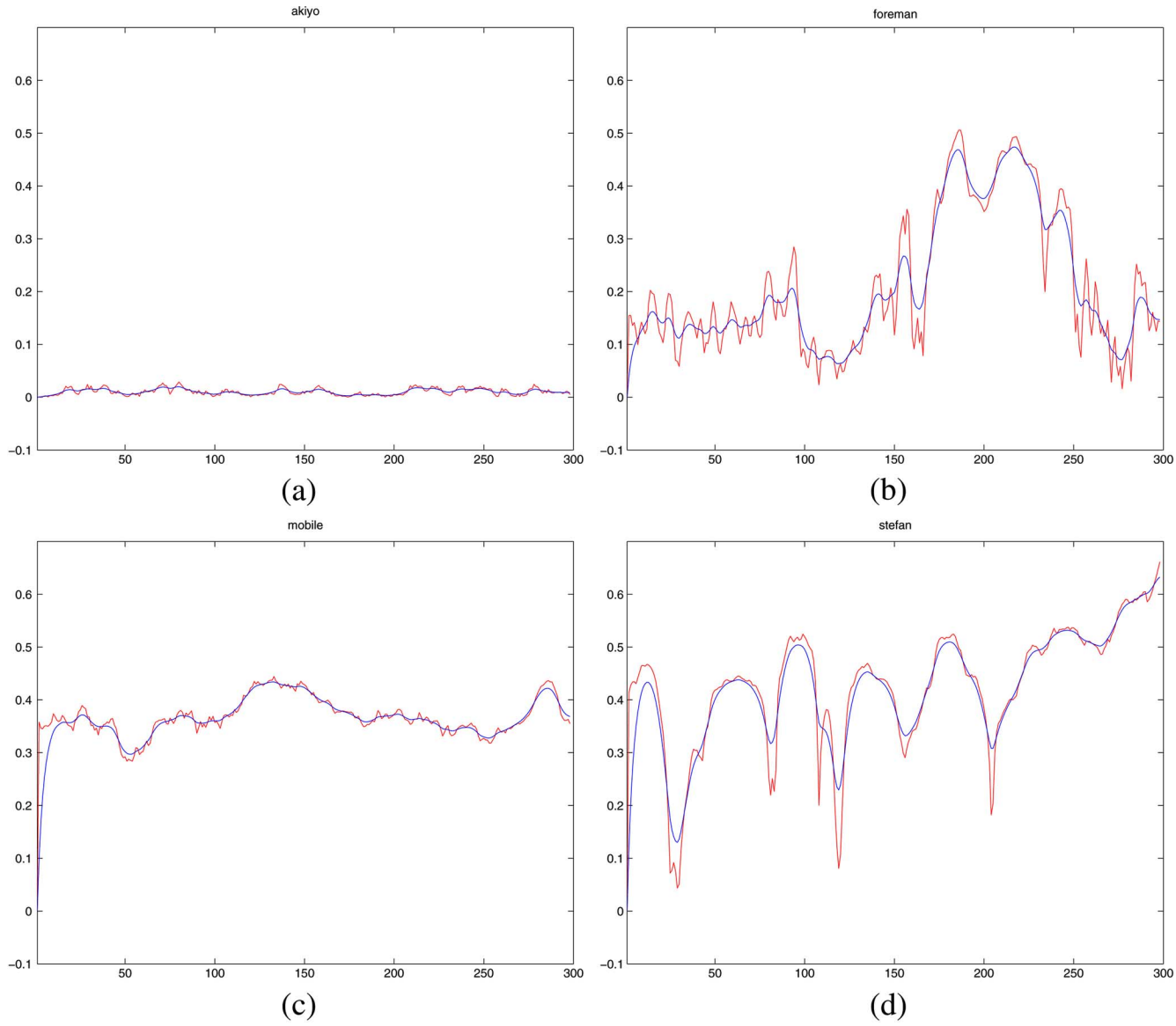


Fig. 3. Amount of motion per frame before and after the filtering for (a) Akiyo, (b) Foreman, (c) Mobile, and (d) Stefan.

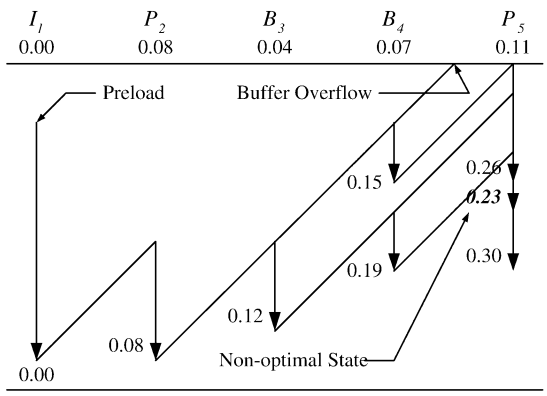


Fig. 4. Discrete-time model for client buffer management. Each buffer fullness level at which an arrow points is considered to be a state. All columns of states form a trellis.

Depending on the tolerable startup delay, preload can vary in the range of $[0, B]$.

Let M_i^j denote the accumulated motion metrics at b_i^j. For state transition from b_p^k to b_q^l, $M_q^l = M_p^k + m_q$, where m_q is the motion metric associated with frame q. If another state b_s^t also leads to state b_q^l, we resolve the "collision" with

$$M_q^l = \max\left\{M_q^l,\ M_s^t + m_q\right\}.$$

Clearly, we always try to maximize the accumulated motion metrics at each state. In case of a "tie," when $M_q^l = M_s^t + m_q$, we can arbitrarily choose one path without affecting global optimality. Alternatively, we may want to choose the one that selects more frames as an additional metric. In summary, a state can be completely characterized by a five-tuple vector $(f_i, m_i, b_i^j, M_i^j, p_i^j)$, where p_i^j is a pointer pointing back to the state that creates the current one and is used for back tracing the path at the final stage.

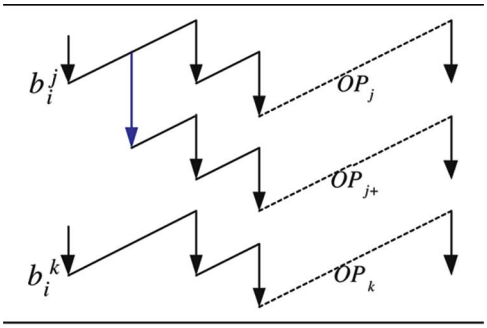


Fig. 5. Illustration of a nonoptimal state.

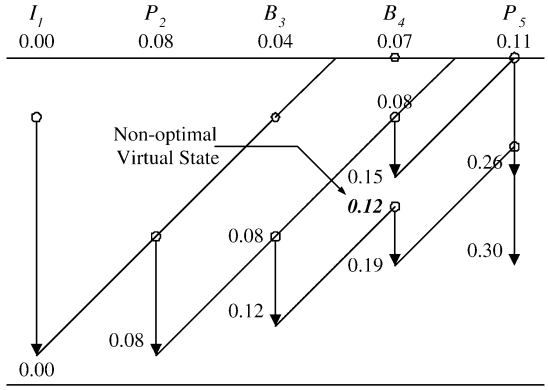


Fig. 6. Illustration of virtual states that facilitate frame-to-frame state transition.

B. Nonoptimal States

Theoretically, the number of possible states increase in an exponential manner with the increase of frame number, which makes the dynamic programming algorithm computationally prohibitive. However, in this research, we find the number of states at each frame can be largely reduced by three factors:

1) buffer size because no state can fall outside the given buffer range;
2) inter-frame dependencies because P- and B-frames need their reference frames for proper decoding;
3) nonoptimal states described below.

Lemma 1: For any two optimal states b_i^j and b_i^k at frame i (an optimal state means a state that could be included in the final optimal path), if $b_i^k < b_i^j$, then $M_i^k \geq M_i^j$, and vice versa. In other words, for optimal states at frame i, M_i^j increases monotonically as b_i^j decreases.

Proof: Suppose $b_i^k < b_i^j$ and $M_i^k < M_i^j$, and b_i^k is on the final optimal frame-selection path OP_k (see Fig. 5). Obviously, b_i^j and b_i^k are mutually exclusive because frame i can only be selected once. Because b_i^j is at a higher buffer state, the least b_i^j can do is to select the same frames that b_i^k has selected from frame i until the end. The accumulated motion values of the new path OP_j is larger than that of OP_k by $M_i^j - M_i^k$. In fact, b_i^j has a better chance to select additional frames such as path OP_{j+}. Whatever the case is, b_i^j can make the accumulated motion values larger by at least $M_i^j - M_i^k$, which contradicts the claim that OP_k is the optimal path. Hence, b_i^k can never be an optimal state. For example, the state b_5^2 at P_5 with $M_5^2 = 0.23$ in Fig. 4 is a nonoptimal state. ■

The elimination of nonoptimal states can dramatically reduce the computation complexity because it not only reduces the number of states at each individual frame but also prevents those nonoptimal states from propagating into subsequent frames.

C. Virtual States

Referring to Fig. 4, we need to consider P_2, B_3, and B_4 in order to obtain all possible states at P_5. This works fine for a small set of frames within a GOP. However, it is not suitable for I-frames in a long video sequence with numerous GOPs since any state transition to a I-frame from any previous frame is allowed. The number of possible state transitions to examine for the I-frame in the next GOP can be potentially huge especially when the I-frame is near to the end of the video sequence. In order to avoid this inconvenient multistep "look back," we introduce the concept of "virtual state." A virtual state at a frame i is a state where a frame selection path passes through the frame i time point without selecting frame i, shown as empty endpoints in Fig. 6. With virtual states, we only need to look back the states at frame i to get all possible states for frame $i+1$.

In particular, a state j at frame i is carried forward and becomes the jth virtual state at frame $i+1$ with the following relations:

$$b_{i+1}^{jv} = \min\left\{b_i^j + \text{bandwidth/framerate},\ B\right\} \tag{8}$$

$$M_{i+1}^{jv} = M_i^j \tag{9}$$

$$p_{i+1}^{jv} = \begin{cases} p_i^j, & \text{if } b_i^j \text{ is a virtual state} \\ \text{pointer to } b_i^j, & \text{if } b_i^j \text{ is an actual state.} \end{cases} \tag{10}$$

From the virtual states, we can easily find the corresponding actual states by

$$\begin{aligned} b_{i+1}^{ja} &= b_{i+1}^{jv} - f_{i+1} \\ M_{i+1}^{ja} &= M_{i+1}^{jv} + m_{i+1} \\ p_{i+1}^{ja} &= p_{i+1}^{jv} \end{aligned} \tag{11}$$

while taking into account inter-frame dependencies. For example, in Fig. 6, the first virtual state at B_3 points back to the actual state at I_1, and hence it cannot create an actual state at B_3 because P_2 is not selected. However, this virtual state cannot be discarded because it might create an optimal state at future I-frames. After obtaining all of the virtual and actual states at frame $i+1$, they are jointly verified for state optimality. In other words, states at frame $i+1$, virtual or actual, must all satisfy Lemma 1. For instance, in Fig. 6, the nonoptimal virtual state b_4^{3v} is removed from B_4 since $(M_4^{3v} = 0.12) < (M_4^{1a} = 0.15)$. Note that eliminating b_4^{3v} also prevents it from propagating to P_5.

Special actions need to be taken when we apply Lemma 1 for I-frames. Consider the following scenario:

$$\ldots\ P_{i-3}\ B_{i-2}\ B_{i-1}\ I_i\ B_{i+1}\ B_{i+2}\ P_{i+3}\ \ldots$$

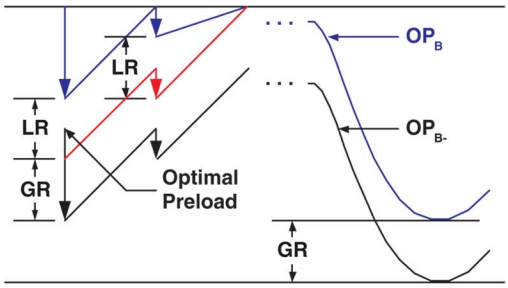


Fig. 7. Obtaining the optimal preload through global and local reductions on the optimal path.

We divide all of the I-frame states into two sets: set 1 contains the states pointing back to an actual state before P_{i-3}, and set 2 contains the states pointing back to P_{i-3}, B_{i-2}, or B_{i-1}. Obviously, only the actual states in set 2 are allowed to create actual states in B_{i+1} and B_{i+2}. If we directly apply Lemma 1 to all the states in both sets 1 and 2, it is possible that an actual state from set 2 is eliminated by a state from set 1. However, the discarded state might create a potentially optimal state at B_{i+1} or B_{i+2}. Therefore, for I-frames, we treat the two sets separately and apply Lemma 1 within each set.

D. Optimal Preload

In the previous subsections, we have shown how to obtain the optimal result given bandwidth, buffer size, and preload. In this subsection, we study the case where the preload is not fixed. It is clear that, for different preload values, the optimal results will be different. Depending on the client's tolerable startup delay, the preload can vary in the range of $[0,\ B]$. Intuitively, a larger preload should yield a larger or at least equal optimal result. The question is: given bandwidth and buffer constraints, what is the minimum preload required to obtain the maximal global optimal result? In other words, there exists a certain preload level, above which we can not get a better optimal result.

Obviously, preload $= B$ is able to give the maximal global optimal result. But the problem is how to bring the preload down to the optimal level. In this research, we propose a two-step approach to bring down the preload level from B. In particular, in the first step we perform global reduction (GR). GR is defined as the distance between the lowest state on the maximal optimal path and the empty buffer line (Fig. 7). It is clear that we can bring the entire maximal optimal path down by GR without changing the global optimal result. In the second step, we perform local reduction (LR). The idea of LR comes from the observation that in the cases of buffer overflow we have to waste some channel bandwidth. In fact, through reducing the preload level, we can avoid wasting channel bandwidth or reduce the amount of bandwidth wasted. Consider two adjacent states b_i^j and b_k^l on the maximal optimal path, where b_k^l is a full buffer state. If buffer overflow occurs during the state transition from b_i^j to b_k^l, the amount of LR at frame k is defined as

$$\mathrm{LR}_k = \min\left\{b_i^j + (k-i)\ (\text{bandwidth/framerate}) - B, b_s^t\right\}$$

where b_s^t is the lowest state on the maximal optimal path up to frame k. It is clear that we can bring down the optimal path from the first frame to frame k by LR_k without affecting the global optimal result. Note that this local reduction has no effect on the optimal path after frame k. Therefore, by jointly applying GR and applying LR at the places of buffer overflow, we are able to bring down the preload to the optimal level.

V. RFS for CBR Channels

In this section, we propose an RFS scheme, which selects frames starting from the last frame of a video sequence until its beginning, to solve the problem of video streaming over CBR channels.

As shown in Fig. 8(a), the symbols above the full buffer line tell the frame type (I, P, or B) as well as its frame number in the sequence. The real numbers below the empty buffer line are the motion metrics associated with each frame. It is obvious that after the client consumes the last frame, the buffer should become empty. Hence, the first state at the last frame is positioned at the empty buffer line, which we refer to as an empty buffer state. A state is represented by the starting point of an arrow, and all arrows are pointing upward because we can record the accumulated motion metrics only if the arrow end is within the full buffer line or its "buffer resource need" can be satisfied. An upward arrow actually means consumption of the buffer data. In contrast, reception of data during each frame slot is reflected by the downward slanted lines between frames. In the case of "buffer underflow," such as that from B_N to P_{N-1}, an empty buffer state will be created. This means that the amount of data transmitted during the period is more than enough to reach the current state, and the server needs to stay idle for some time or reduce the transmission rate. Because an path can terminate at any frame, if there is no empty buffer state at a frame, we will create one such as that at B_{N-2}.

Fig. 8(a) is not intuitive to interpret. We use a simple "buffer mirroring" technique to make the computation easier and more intuitive. Imagine that there is a mirror aligned with the empty buffer line in Fig. 8(a). If we look from the full buffer line side, we shall see a mirrored buffer model as shown in Fig. 8(b). The buffer mirroring effect makes the computation just like what we do in the forward frame-selection scheme except that the selection is from the end of the video sequence to the beginning. All of the concepts, including Lemma 1, discussed for forward frame selection can also be applied to the mirrored buffer model.

Note that, at the first frame, RFS generates multiple accumulated motion metric gains at different states and each gain corresponds to the optimal result that we can get at that preload level. In other words, RFS runs only once and gets all the optimal results for different start-up delays, which is very useful for the case of multiple clients with different start-up delay requirements. For those preloads that are not exactly matched in the RFS results, we use the results of their nearby lower matched preloads. On the contrary, the forward frame selection scheme has to run many times in order to obtain all the optimal results for multiple start-up delay requirements, which is very time-consuming.

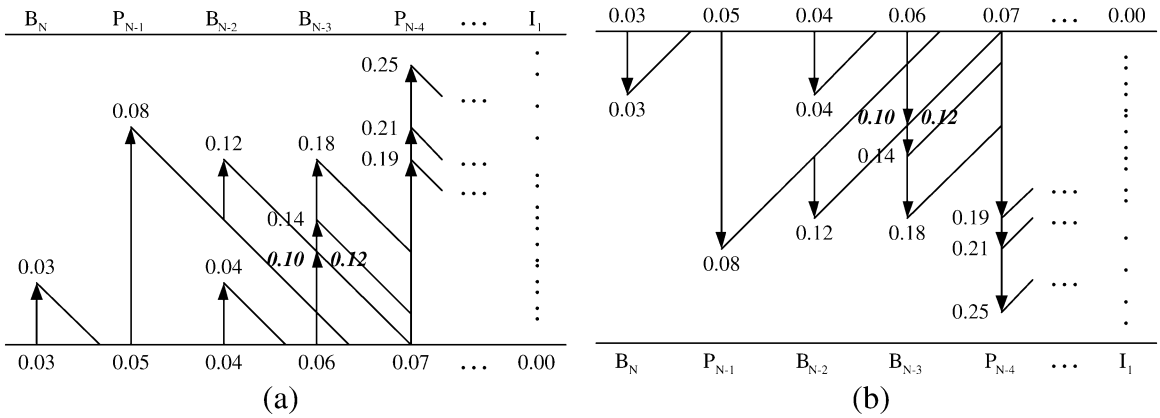


Fig. 8. Reverse Viterbi selection (a) in normal mode and (b) with buffer mirroring.

VI. RFS for VBR Channels

For the VBR channels (piecewise constant channels), the optimal frame selection becomes extremely difficult since we do not know when and how the channel is going to change in the future., i.e., the channel variation is unpredictable. Suppose we know the range of bandwidth variation; one possible approach is to offline compute the optimal frame selection path according to the middle value of the bandwidth variation range. Note that this middle value is not the average bandwidth, which we do not know. We can also compute the optimal path according to the minimum (or maximal) bandwidth, but it will lead to high channel bandwidth wasting (or high occurrence of buffer underflow). During transmission, the JIT algorithm [14] is applied for on-the-fly adaptation in response to bandwidth changes. If the current bandwidth is larger, JIT raises buffer occupancy, which reduces the probability of future buffer underflow. In case of buffer underflow, JIT drops the current frame right the way. When overflow occurs, JIT reduces the transmission rate. Clearly, all of these approaches use the precomputed frame selection path and the actual path cannot be better than the precomputed one. Another possible approach is to use JIT directly without any precomputed frame-selection path, which tries to send all of the frames without content awareness.

In this paper, we propose to use the RFS scheme for video streaming over VBR channels. In particular, we sample the bandwidth variation range into a finite sequence of channel rates. For a given client buffer size, we run the RFS scheme for each sampled channel rate. During transmission, if the starting buffer occupancy status is b_1 and the current bandwidth is C, we will first classify it into one of the preselected channel rates C_1 and then retrieve the optimal frame path for the channel rate C_1 with a starting state of b_1. If at frame N_2 the buffer state is b_2 and the bandwidth is changed to C_2, we will retrieve the optimal frame path starting at frame N_2 for the channel rate C_2 with a starting state of b_2. Fig. 9 shows such an example. In this way, the global optimality is approximately preserved under dynamic changing network conditions. The key advantage here is that we only need to run RFS k times, where k is the number of channel bandwidth samples, and it can apply to any pattern of piecewise constant channels occurring at any time as long as the changes are within the variation range.

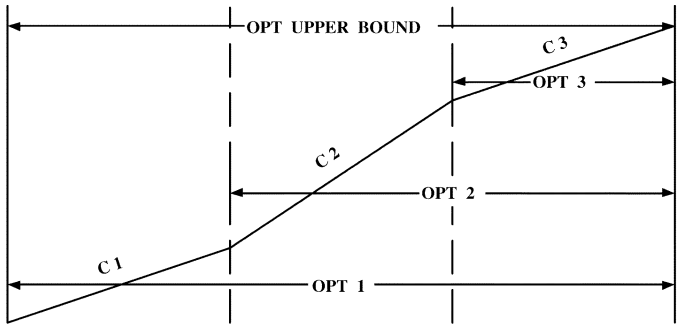


Fig. 9. Optimal path switching for video streaming over VBR channels.

TABLE I
PROPERTIES OF THE FOUR MPEG-4 VIDEO TRACES

Video Title	akiyo	foreman	mobile	stefan
Bitrate(kbps)	47.52	288.25	745.10	977.00
Motion Metric	3.05	63.2	109.28	127.0
Buffer size (kbytes)	9.6	45	150	150
Bandwidth Range (kbps)	[15, 52.5]	[120, 300]	[420, 780]	[330, 1050]

VII. Experimental Results

A. Experimental Results of Short Videos

We conduct experiments to compare six algorithms: "OFS," "OFS + OP," "Z-B," "JIT," "Greedy," and "Greedy + WR," where "OFS" stands for our proposed forward optimal frame-selection algorithm without optimal preload, "OFS + OP" stands for our proposed forward optimal frame selection with optimal preload, "Z-B" represents the Z-B diagram algorithm with frame stuffing, "Greedy" represents the greedy algorithm with weighted motion metric, and "Greedy + WR" represents the greedy algorithm with weighted ratio metric m_i^r. Note that we have also obtained the results of the reverse Viterbi algorithm. Since they are the same as "OFS" and "OFS + OP" for the respective cases, we do not list them out for the conciseness of this paper.

We choose four representative MPEG-4 CIF video traces to evaluate the performance of the various algorithms. Each trace contains 300 frames with a frame rate of 30 frames/s. They are both encoded in the pattern: $\ldots I\,B\,B\,P\,B\,B\,P\ldots$ with a GOP

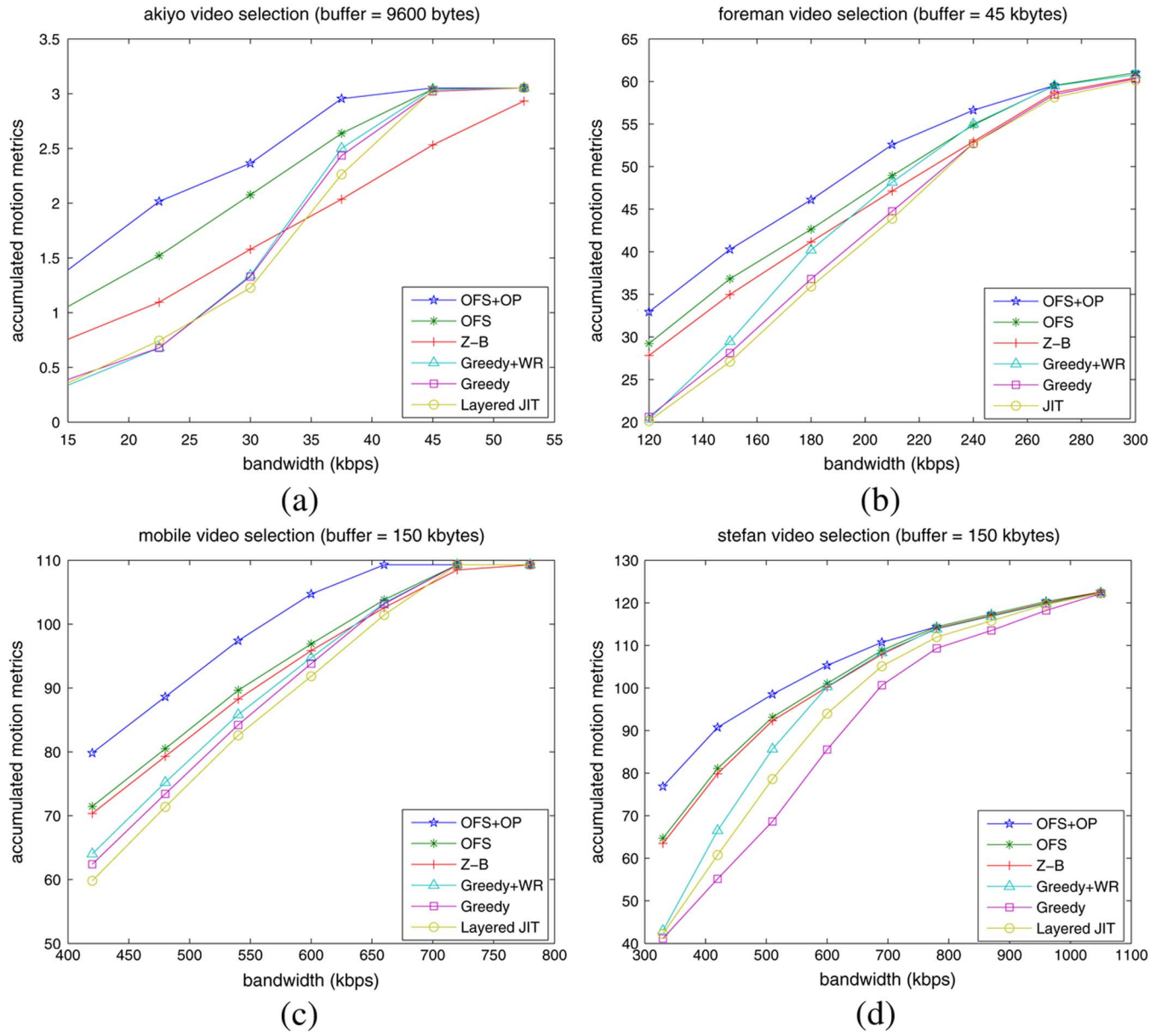


Fig. 10. Selected motion metric gains under different algorithms for (a) Akiyo, (b) Foreman, (c) Mobile, and (d) Stefan.

size of 90. The Akiyo video trace contains little motion information and is coded at very lower bit rate. The Foreman video trace containing moderate motion is coded at a relatively lower bit rate. The Mobile video trace containing nearly flat high motion and complex texture is coded at high bit rate. The Stefan containing the highest motion information is coded with the highest average bit rate (see Table I). The table also shows the setting of the buffer sizes and the bandwidth ranges for each video sequence. Note that, except for the "OFS + OP," all of the other algorithms use fixed preloads, i.e., half of the buffer sizes.

Fig. 10 shows the frame-selection results using the six different algorithms under different channel bandwidth. It can be seen that the "OFS + OP" outperforms all of the other algorithms especially at low bandwidth regions since it fully explores the buffer capacity. Under the fixed preload levels, our proposed "OFS" algorithm always gives the optimal results. For the "Z-B" algorithm, we choose step size $= 100$ bytes for all the video traces. The "Z-B" performance of Stefan is only slightly worse than "OFS" while the gap is very large for Akiyo. This is because Stefan has a very high bit rate and the stuffed data only occupies a small percentage of the total bandwidth whereas for Akiyo the stuffed data severely degrades bandwidth utilization. As expected, the selection results of the "Greedy + WR" algorithm are very close to the optimal results except at low bandwidth regions. This is because, at low bandwidth, some P-frames that have small motion metrics still tend to be selected due to the large weights assigned to them. As a result, the "Greedy + WR" performs as poor as the "Greedy" algorithm and the "JIT" algorithm at low bandwidth regions. Note that for the "JIT" algorithm I- and P-frames are always considered first, which is equivalent to assign them "weights." It is surprising that the JIT that has no content awareness performs

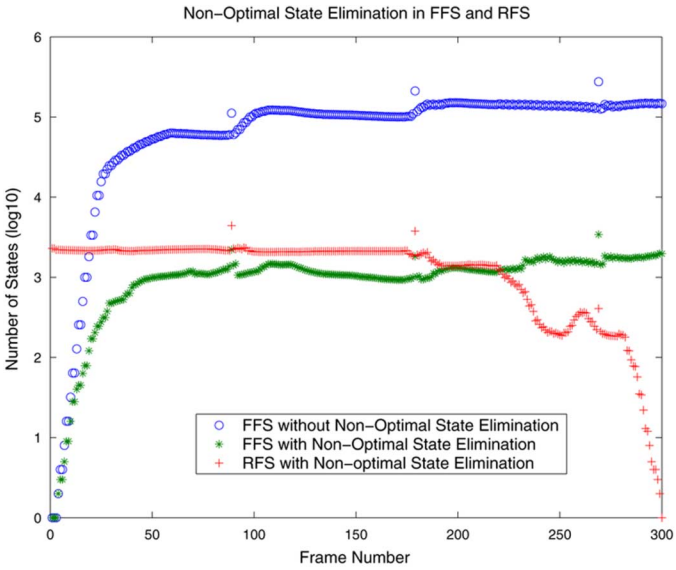


Fig. 11. Number of states at each frame.

better than the "Greedy" algorithm for Stefan. The reason is perhaps that for Stefan high-motion frames have very large frame sizes and the "Greedy" algorithm completely ignores the cost of consuming large frames.

Fig. 11 shows the number of states at each frame for Stefan using the "OFS" algorithm with and without the nonoptimal state elimination, and the "Reverse Viterbi" algorithm. The bandwidth is 600 kb/s and the buffer size is 150 kbytes. For "OFS," the preload is set to 75 kbytes. As shown in the figure, the initial linear increment indicates an exponential growth of states with the increase of the frame number (note that the log-scale is used at the vertical axis). The curves then become relatively constant because of the buffer size constraint. Those discontinuous points are mainly due to inter-frame dependencies. Comparing the cases with and without the nonoptimal state elimination, we can see that there exist a huge number of nonoptimal states. By applying the proposed nonoptimal state elimination, we reduce the number of states at each frame by approximately 100 times. In addition, we can observe that the number of states for the "Reverse Viterbi" algorithm is more or less the same as that for the "OFS."

B. Experimental Results of Long Video

The purpose of the previous experiments is to prove the concepts of our proposed algorithms, where the short test video sequences and the small client buffer are being used. However, in practical applications such as VoD, the video is typically much longer and the buffer size even in today's mobile devices can be much bigger. Therefore, in this section, we study the performance of our proposed algorithms in the cases of long video and large client buffer size.

We create a longer video sequence, where we equally choose 15 times of each of the four video traces in Table I and randomly shuffle these sixty 300-frame traces. The generated mixed sequence is encoded into MPEG-4 bitstream with an average bitrate of 542.33 kb/s, a pattern of IPBBPBBP…, a GOP size of 60, and an accumulated motion values of 4598.21. The buffer size we consider is in the range from 256 to 2048 kbyte. For such a long video sequence and large buffer size, with only nonoptimal state elimination, the computational complexity is still too high. Thus, in addition to nonoptimal state elimination, frame stuffing is used to further reduce complexity at the cost of sacrificing some bandwidth resources. Note that, in the following, we use our proposed RFS scheme for experiments due to its low complexity and flexibilities, and hereafter "OFS" stands for the proposed RFS scheme.

1) Impact of Preload and Frame Stuffing: Fig. 12 shows the optimal accumulated motion values that we can achieve under different frame stuffing sizes and preloads with a fixed bandwidth of 300 kb/s. It is obvious that the smaller the stuffing size we use, the better performance we achieve since less bandwidth is wasted. Comparing Fig. 12(a) and (b), we find that large preloads for the smaller buffer do not lead to as proportionate gains in the accumulated motion metrics as those for a larger buffer. In addition, in the case of a 512-kbyte buffer and the frame stuffing size of 1 byte (i.e., no stuffing), the accumulated motion results stop at the preload time of a little over 4 s, which is less than half of the largest allowable preload time, 13.65 s $(8 \times 512/300)$. On the contrary, the corresponding results in the case of 1024 kbytes buffer stop around 18 s. The stop point is actually the point of the optimal preload, after which increasing preload will not change the performance. The reason to have a shorter optimal preload for a 512-kbyte buffer is that a smaller buffer is more likely to incur buffer overflow at an early stage, and once buffer overflow occurs, increasing the preload becomes useless (see Section IV-D).

2) Effectiveness of Complexity Reduction: Here, we evaluate the effectiveness of frame stuffing as well as nonoptimal state elimination for reducing the complexity of dynamic programming. Fig. 13 shows the average number of states per frame for different frame-stuffing step sizes under different buffer conditions. As we can see, the number of states per frame reduces with increasing stuffing sizes. For instance, for buffer $= 1024$ kbytes, the number of states reduces from over 128 k (2^{17}) at stuffing $= 1$ byte (i.e., no stuffing) down to a little over 4 k (2^{12}) at stuffing $= 200$ bytes, a dramatic reduction of 32 times. However, as the number of states becomes less, further reduction by increasing the stuffing size appears to be less significant.

The effectiveness of nonoptimal state elimination can also be evaluated from Fig. 13. In particular, let 2^T represent the theoretical number of states per frame after taking the contribution of frame stuffing into account. For example, for buffer $=$ 512 kbytes and stuffing $= 200$ bytes, the theoretical value is $2^T = (512\text{ k}/200) = 2^{11.36}$. Let 2^R represent the recorded average number of states in Fig. 13. It is clear that the percentage calculated by $(2^T - 2^R)/2^T$ indicates the contribution from nonoptimal state elimination. Tables II and III show this percentage of state reduction due to nonoptimal state elimination at different buffer sizes. It can be seen that with no frame stuffing the reduction percentage can be as high as 95.79% for buffer $= 512$ kbytes. However, as we increase the stuffing size, the reduction becomes less effective. This is not surprising because frame states are spaced out by at least a distance equal to the stuffing size. As the stuffing size increases, it becomes less likely to create nonoptimal states. Comparing Tables II and III,

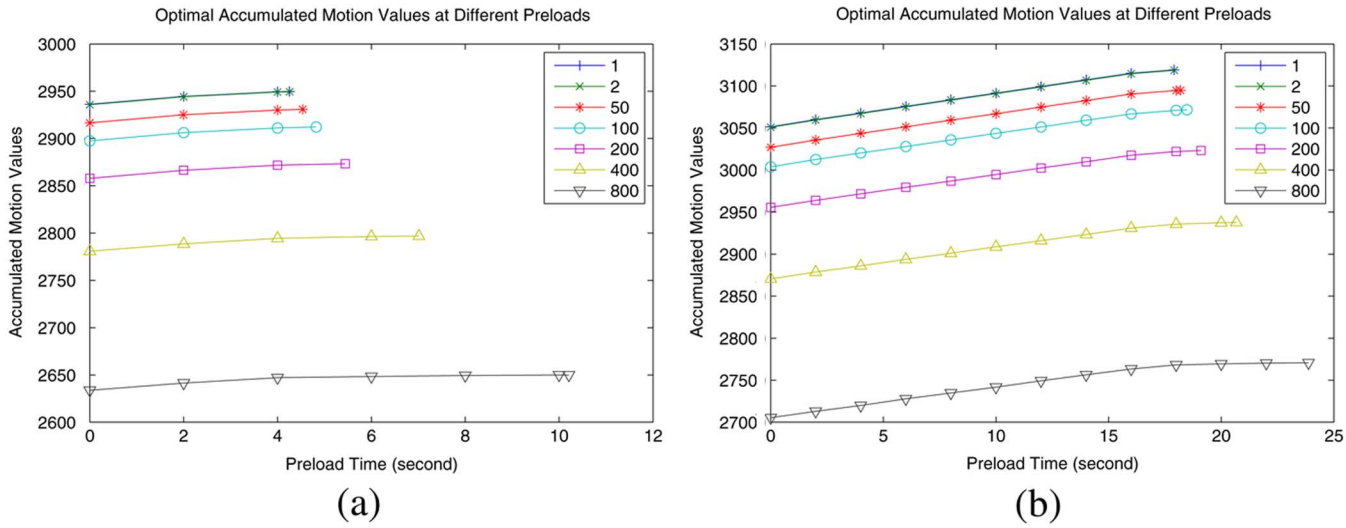


Fig. 12. Results of the accumulated motion metrics for delivering the long VBR video under different frame stuffing sizes and preloads with a buffer size of (a) 512 kbytes and (b) 1024 kbytes.

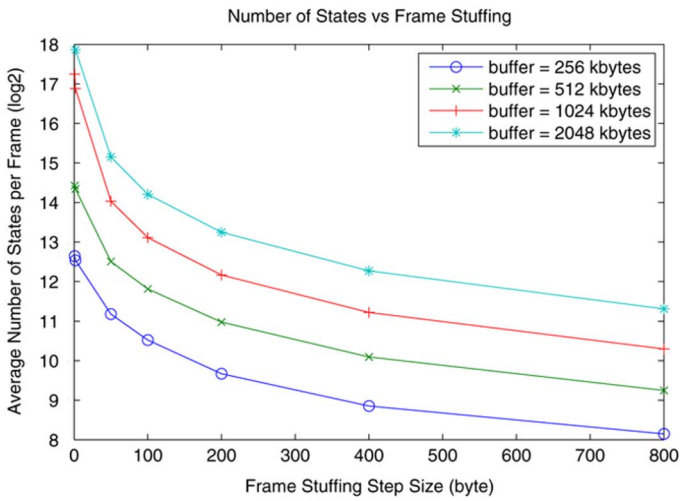


Fig. 13. Frame stuffing versus average number of states per frame with bandwidth $= 300$ kb/s.

TABLE II
EFFECTIVENESS OF NONOPTIMAL STATE ELIMINATION AT BUFFER $= 512$ kbytes

Number of States per Frame ($\log_2$)	Frame Stuffing Step Size (byte)						
	1	2	50	100	200	400	800
Theory	19	18	13.36	12.36	11.36	10.36	9.36
Record	14.43	14.34	12.51	11.81	10.98	10.09	9.25
Reduction (%)	95.79	92.09	44.52	31.70	23.16	17.07	7.34

TABLE III
EFFECTIVENESS OF NON-OPTIMAL STATE ELIMINATION AT BUFFER $= 1024$ kbytes

Number of States per Frame ($\log_2$)	Frame Stuffing Step Size (byte)						
	1	2	50	100	200	400	800
Theory	20	19	14.36	13.36	12.36	11.36	10.36
Record	17.25	16.88	14.03	13.11	12.17	11.22	10.30
Reduction (%)	85.13	77.00	20.45	15.91	12.34	9.25	4.07

we can conclude that a larger buffer creates a smaller percentage of nonoptimal states, and nonoptimal state elimination is more effective for small stuffing sizes and small buffers.

Note that, although the number of states after frame stuffing and nonoptimal state elimination is still large, we find that at each frame many consecutive states point to the same frame to be selected next. Therefore, in our implementation, we group those states leading to the same next destination together and only store the ranges of different groups. In this way, the resulted storage overhead is actually not much.

3) Performance Comparison: Fig. 14(a) shows the results of the accumulated motion metrics of different frame selection algorithms under different bandwidth conditions, where "OFS + n" refers to our proposed optimal RFS algorithm with n bytes of frame stuffing. The observations are similar to those described in Section VII-A. Fig. 14(b) shows the results under different buffers. It can be seen that all the OFS + n algorithms outperform the other three algorithms under all the buffer conditions. We can also observe that a larger buffer such as 2048 kbytes does not yield significant gain. This is due to the bandwidth constraint. In addition, it is interesting to see that the Greedy algorithm has a worse performance when the buffer increases from 1024 to 2048 kbytes. The reason is perhaps that a larger buffer allows the Greedy algorithm to select more large-size frames at the beginning, which consumes most of the bandwidth and thus compromises the overall gain. Note that we did not compare with the Z-B diagram algorithm since its accumulated motion results are the same as those for our proposed OFS at the same stuffing size.

4) Performance of VBR Channels: We use piecewise-CBR channels to approximate the bandwidth variations of VBR channels. Particularly, we divide the time into consecutive T-second intervals and at the beginning of each interval the bandwidth is randomly chosen from the set: $\{100, 200, 300, 400, 500\}$ kbps. The time interval T are set to 2 and 10 seconds, representing a fast-changing VBR channel and a slow-changing channel, respectively.

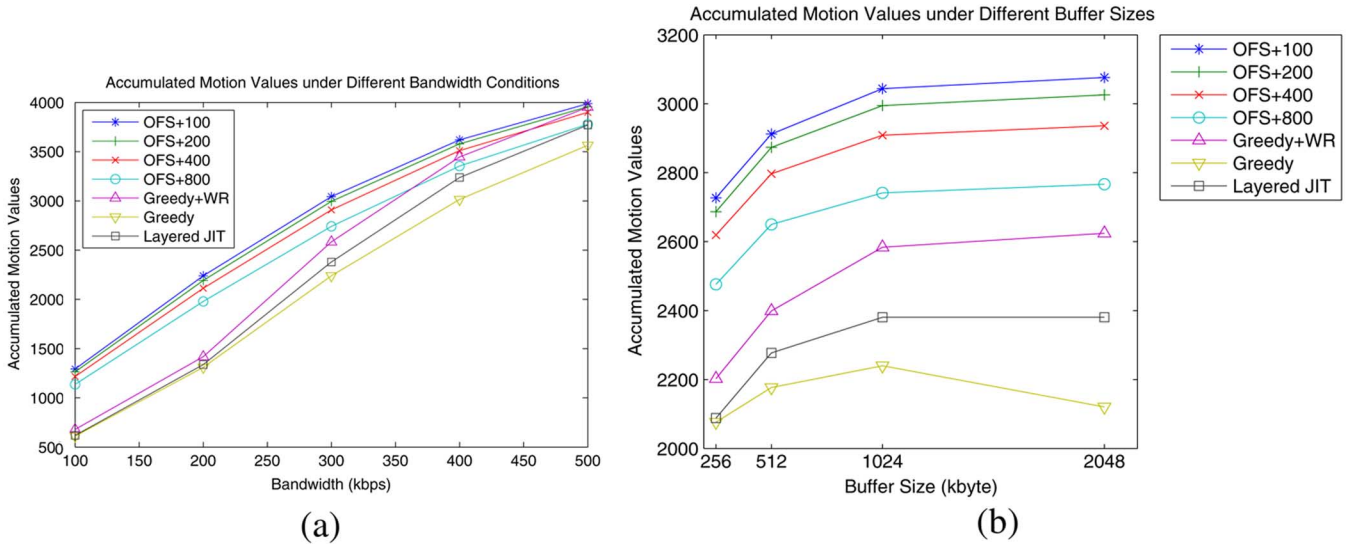


Fig. 14. Results of the accumulated motion metrics of different frame selection algorithms with a fixed preload of 375 kbytes (300 kb/s $\times$ 10 s). (a) Under different bandwidth conditions with buffer $=$ 1024 kbytes (b) Under different buffers with bandwidth $=$ 300 kb/s.

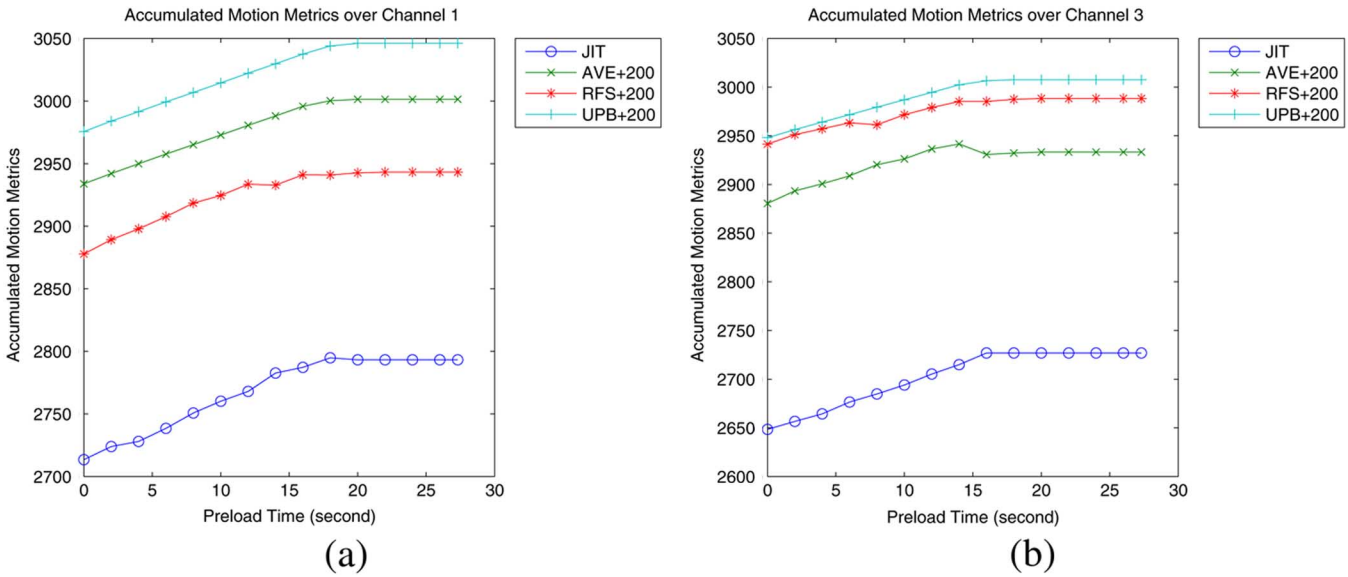


Fig. 15. Results of the accumulated motion metrics with buffer $=$ 1024 kbytes. (a) Under the fast-changing VBR channel. (b) Under the slow-changing VBR channel.

Fig. 15 shows the frame selection results of different algorithms over the VBR channels, where "Ave" is the algorithm using the middle bandwidth to compute the optimal path (see Section VI), and "UPB" is the upper bound that achieves the global optimization by assuming the channel bandwidth variation is known *a priori*. It can be seen that both AVE $+$ 200 and OFS $+$ 200 outperform JIT significantly. For the case of the fast-changing VBR channel in Fig. 15(a), AVE $+$ 200 has a better performance than OFS+200. This is because OFS is optimal on the condition that the new bandwidth will remain constant until the end of the sequence. With the bandwidth varies so frequently, the global optimality of the OFS is severely deviated. On the contrary, for the case of the slow-changing VBR channel in Fig. 15(b), OFS $+$ 200 outperforms AVE $+$ 200. This is because, with less frequent bandwidth changes, the OFS can better preserve the global optimality over longer CBR channel segments while the AVE $+$ 200 algorithm is severely degraded by the long-term low-bandwidth CBR channel segments.

VIII. Conclusion

In this paper, we have studied the problem of optimal frame selection for streaming stored video over both CBR and VBR channels using dynamic programming. Our major contributions are threefold. First, we have proposed the elimination of nonoptimal states, and combining with the frame stuffing it can effectively reduce the computational complexity of dynamic programming, especially in the cases of small stuffing sizes and small buffers. Second, we have proposed the RFS algorithm, which can find the optimal results for any preload in one round for CBR channels. Third, our proposed RFS algorithm has been

smartly extended for the VBR channels, which can be modeled as piecewise CBR channels. Experimental results have demonstrated that with modest complexity our proposed algorithm achieves much better performance than the common JIT algorithm, especially in the cases of slow-changing VBR channels.

REFERENCES

[1] S. Sen, J. L. Rexford, J. K. Dey, J. F. Kurose, and D. F. Towsley, "Online smoothing of variable-bit-rate streaming video," *IEEE Trans. Multimedia*, vol. 2, no. 1, pp. 37–48, Mar. 2000.

[2] M. Wu, R. A. Joyce, H.-S. Wong, L. Guan, and S.-Y. Kung, "Dynamic resource allocation via video content and short-term traffic statistics," *IEEE Trans. Multimedia*, vol. 3, no. 2, pp. 186–199, Jun. 2001.

[3] M. W. Garrett and W. Willinger, "Analysis modeling and generation of self-similar VBR video traffic," in *Proc. ACM SIGCOMM*, Aug. 1994, vol. 3, no. 2, pp. 269–280.

[4] M. Grossglauser, S. Keshav, and D. N. C. Tse, "RCBR: A simple and efficient service for multiple time-scale traffic," *IEEE/ACM Trans. Netw.*, vol. 5, no. 6, pp. 741–755, Dec. 1997.

[5] T. V. Lakshman, A. Ortega, and A. R. Reibman, "VBR video: Tradeoffs and potentials," *Proc. IEEE*, vol. 86, no. 5, pp. 952–973, May 1998.

[6] A. R. Reibman and B. G. Haskell, "Constraints on variable bit-rate video for ATM networks," *IEEE Trans. Circuits Syst. for Video Technol.*, vol. 2, no. 12, pp. 361–372, Dec. 1992.

[7] W. Feng, F. Jahanian, and S. Sechrest, "An optimal bandwidth allocation strategy for the delivery of compressed prerecorded video," *Multimedia Syst.*, vol. 5, no. 5, pp. 297–309, Sep. 1997.

[8] W. Feng, B. Krishnaswami, and A. Prabhudev, "Proactive buffer management for the streamed delivery of stored video," *ACM Multimedia*, pp. 285–290, Sep. 1998.

[9] J. M. McManus and K. W. Ross, "A dynamic programming methodology for managing prerecorded VBR sources in packet-switched networks," *Telecommun. Syst.*, vol. 9, no. 2, pp. 223–247, 1998.

[10] J. Rexford and D. Towsley, "Smoothing variable-bit-rate video in an internetwork," *IEEE/ACM Trans. Netw.*, vol. 7, no. 2, pp. 202–215, Apr. 1999.

[11] W. Feng and J. Rexfordy, "A comparison of bandwidth smoothing techniques for the transmission of prerecorded compressed video," in *Proc. IEEE INFOCOM*, Apr. 1997, pp. 58–66.

[12] 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," *IEEE/ACM Trans. Netw.*, vol. 6, no. 4, pp. 397–410, Aug. 1996.

[13] W. Feng, "Rate-constrained bandwidth smoothing for delivery of stored video," *SPIE Multimedia Computing Netw.*, pp. 316–327, Feb. 1997.

[14] Z.-L. Zhang, S. Nelakuditi, R. Aggarwal, and R. P. Tsang, "Efficient selective frame discard algorithms for stored video delivery across resource constrained networks," in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 472–479.

[15] J. K.-Y. Ng and S. Song, "A video smoothing algorithm for transmitting MPEG video over limited bandwidth," in *Proc. 4th Int. Workshop Real-Time Computing Syst. Applic.*, Oct. 1997, pp. 229–236.

[16] X. S. Zhou and S.-P. Liou, "Optimal nonlinear sampling for video streaming at low bit rates," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 6, pp. 535–544, Jun. 2002.

[17] W. C. Feng and M. Liu, "Extending critical bandwidth allocation techniques for stored video delivery across best-effort networks," *Int. J. Commun. Syst.*, vol. 14, pp. 925–940, Sep. 2001.

[18] T. Gan, K. K. Ma, and L. Zhang, "Dual-plan bandwidth smoothing for layer-encoded video," *IEEE Trans. Multimedia*, vol. 7, pp. 379–392, Apr. 2005.

[19] P. A. Chou and Z. Miao, "Rate-distortion optimized streaming of packetized media," *Microsoft Res. Tech. Rep.*, Feb. 2001.

[20] J. Chakareski and P. A. Chou, "Application layer error-correction codng for rate-distortion optimized streaming to wireless clients," *IEEE Trans. Commun.*, vol. 52, pp. 1675–1687, Oct. 2004.

[21] J. Chakareski, S. Han, and B. Girod, "Layered coding versus multiple descriptions for video streaming over multiple paths," *Multimedia Syst.*, vol. 10, pp. 275–285, Jan. 2005.

[22] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "TCP friendly rate control TFRC: Protocol specification," *RFC 3448, Internet Engineering Task Force*, Jan. 2003.

[23] Y. Ma and H. J. Zhang, "A new perceived motion based shot content representation," in *Proc. IEEE ICIP*, 2001, vol. 3, pp. 426–429.

[24] H. Yi, D. Rajan, and L.-T. Chia, "Global motion compensated key frame extraction from compressed videos," in *Proc. IEEE ICASSP*, Mar. 2005, pp. 453–456.

[25] H. Yi, D. Rajan, and L.-T. Chia, "A new motion histogram to index motion content in video segment," *Pattern Recogn. Lett.*, vol. 26, no. 9, pp. 1221–1231, Jul. 2005.

[26] A. Tekalp, *Digital Video Processing*. Englewood Cliffs, NJ: Prentice-Hall, Aug. 1995.

[27] E. Spelke, R. Kestenbaum, D. Simons, and D. Wein, "Spatiotemporal continuity, smoothness of motion and object identity in infancy," *Brit. J. Development. Psychol.*, vol. 13, pp. 113–142, 1995.

[28] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.

Dayong Tao received the B.E. degree in electrical and electronic engineering from Nanyang Technological University (NTU), Singapore, in 2004. He is currently working toward the M.S. degree at the School of Computer Engineering, NTU.

His research interests include image processing, video coding and multimedia networking.

Jianfei Cai (S'98–M'02) received the Ph.D. degree from the University of Missouri-Columbia in 2002.

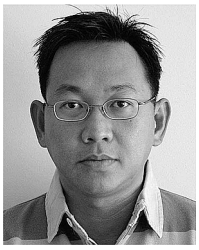
Currently, he is an Assistant Professor with Nanyang Technological University, Singapore. His major research interests include digital media processing, multimedia compression, communications, and networking technologies. He has published more than 50 technical papers in international conferences and journals. He has been actively participated in program committees of various conferences, and he is the mobile multimedia track co-chair for ICME 2006, the technical program co-chair for Multimedia Modeling (MMM) 2007 and the conference co-chair for Multimedia on Mobile Devices 2007. He is also an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY.

Haoran Yi received the B.S. degree in electrical and information engineering from Huazhong University of Science and Technology, Wuhan, China, in 2002. He is currently working toward the Ph.D. degree at the School of Computer Engineering, Nanyang Technological University, Singapore.

His research interests include content-based video analysis and representation, image understanding, and other issues concerning image and video technology.

Deepu Rajan received the B.E. degree in electronics and communication engineering from the Birla Institute of Technology, Ranchi, India, the M.S. degree in electrical engineering from Clemson University, Clemson, AL, and the Ph.D. degree from Indian Institute of Technology, Bombay.

From April 1992 until May 2002, he was a Lecturer with the Department of Electronics, Cochin University of Science and Technology, India. Since June 2002, he has been an Assistant Professor with the School of Computer Engineering, Nanyang Technological University, Singapore. His research interests include image and video processing, computer vision, and multimedia signal processing.



Liang-Tien Chia received the B.S. and Ph.D. degrees from the Loughborough University of Technology, Louborough, U.K., in 1990 and 1994, respectively.

He is the Director of the Centre of Multimedia and Network Technology and an Associate Professor WITH the Division of Computer Communications, School of Computer Engineering, Nanyang Technological University, Singapore. His research interests include image/video processing and coding, multimodal data fusion, multimedia adaptation/transmission and multimedia over the Semantic Web. He has published over 80 research papers.

King N. Ngan (M'79–SM'91–F'00) received the Ph.D. degree in electrical engineering from the Loughborough University of Technology, Loughborough, U.K.

He is currently a Chair Professor with the Department of Electronic Engineering, Chinese University of Hong Kong, Hong Kong, and was previously a Full Professor with the Nanyang Technological University, Singapore, and the University of Western Australia, Australia. He is an Associate Editor of the *Journal on Visual Communications and Image Representation* as well as an area editor of the *EURASIP Journal of Signal Processing: Image Communication*. He has also served as an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY and the *Journal of Applied Signal Processing*. He chaired a number of prestigious international conferences on video signal processing and communications and served on the advisory and technical committees of numerous professional organizations. He has published extensively including three authored books, five edited volumes, and over 200 refereed technical papers in the areas of image/video coding and communications.

Professor Ngan is a Fellow of the Institute of Electronics Engineers (U.K.) and IEAust (Australia).