Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT

Computer Communications 26 (2003) 639–651

computer communications

www.elsevier.com/locate/comcom

Online variable-bit-rate video traffic smoothing

Guohong Cao[a,*], Wu-chi Feng[b], Mukesh Singhal[c]

[a]Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802, USA
[b]Department of Computer Science and Engineering, Oregon Graduate Institute, Beaverton, OR 97006, USA
[c]Department of Computer Science, The University of Kentucky, Lexington, Ky 40506, USA

Received 8 July 2002; accepted 8 July 2002

Abstract

The efficient transmission of variable-bit-rate (VBR) video streams is complicated by the burstiness that video compression standards such as MPEG introduce. Most of the existing techniques concentrate on stored video traffic smoothing or real-time video traffic smoothing. However, there is a growing number of live video applications, such as video-casts of courses or television news, where many clients may be willing to tolerate a playback delay of several seconds or minutes in exchange for a smaller throughput requirement. Bandwidth smoothing for these live video applications is referred to as *online smoothing*. In this paper, in order to measure the effectiveness of online video smoothing methods, we propose a benchmark algorithm, which provides an upper bound on some of the performance metrics in the smoothing results. Based on this algorithm, we found that significant discrepancy exists between the results produced by the existing online smoothing methods and the upper bound. With this observation, we focus on designing algorithms that can improve the smoothing results. Experimental results show that the proposed algorithms make considerable improvements compared to existing smoothing methods.

Keywords: Bandwidth smoothing; Traffic management; Video-on-demand; Compressed video; Online smoothing

1. Introduction

Many emerging multimedia applications, such as video-on-demand and video-casting, rely on the efficient transfer of stored or live video. Transferring high-quality video requires a large amount of network band-width. Even effective compression techniques, such as MPEG [8] and motion-JPEG [22], still result in video streams with bandwidth requirements in the range of 2–10 megabits/second. In addition, compressed video exhibits significant burstiness on multiple time scales [9,10,18] due to the natural variation within and between scenes, as well as the frame structure of the encoding algorithm [10,13,17,18,20]. This variability in the bandwidth requirements complicates the provisioning of resources along the path from the video source to the client site.

One possible solution to this problem is for the encoder to reduce burstiness by adjusting the quantization level of the frames across time, particularly during scenes with high bandwidth requirements. However, a constant-bit-rate encoding reduces the picture quality, which is likely to be most noticeable to users at scene changes or intervals with significant detail or motion. For the same average bandwidth, a variable-bit-rate (VBR) encoding offers higher quality than a constant-bit-rate encoding [2,12].

The efficient transfer of VBR video requires effective techniques for handling burstiness. Although the server, network, and client could conceivably allocate resources based on the peak rate of the stream, such over-provisioning is extremely wasteful and undermines the benefits of a constant-quality encoding. Alternatively, resources could be allocated based on certain assumptions about how a VBR stream would multiplex with other traffic. Models based on statistical multiplexing, and particularly the theory of effective bandwidth, are useful for network provisioning, particularly on high-bandwidth links that carry a large number of traffic streams. However, statistical multiplexing does not offer deterministic guarantees and is less useful on lower-bandwidth links that multiplex a small or moderate number of streams. Despite rapid increases in backbone capacity in recent years, most broadband access networks cannot carry more than a handful of high-quality video

* Corresponding author.
E-mail addresses: gcao@cse.psu.edu (G. Cao), wuchi@cse.ogi.edu (W. Feng), singhal@cs.uky.edu (M. Singhal).

0140-3664/03/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0140-3664(02)00197-4

streams at a time. Instead of relying on statistical multiplexing to handle burstiness, we believe that it is necessary to reduce the variability of individual video streams.

Stored video applications can reduce the burstiness of the traffic by capitalizing on a priori knowledge of the frame sizes in the compressed video stream. In particular, the server can smooth the stream by prefetching video frames in advance of each burst. By initiating transmission early, the server can send large frames at a lower rate without disrupting the client application. The client system can then retrieve, decode, and display frames at the frame rate. The potential benefit of prefetching depends on the size of the client buffer. The server must limit the amount of prefetching to prevent overflow of this buffer; however, to avoid underflow, the server should transmit enough data to allow the client to drain its buffer at the frame display rate. Given a fixed client buffer size, many bandwidth smoothing algorithms [4,5,7,11,14] have been proposed to minimize the peak bandwidth requirements, while also optimizing other performance metrics, such as the variability of bandwidth requirement, the buffer residency time, and the number of rate changes.

Such a priori information, however, is impossible for real-time video applications such as video conferencing. In these applications, digital video frames are captured and compressed in real-time. The resulting video data are then transmitted over the network to a displaying process no more than a few hundred milliseconds and thus allows only a few frames to be buffered by the transmitter. Thus, real-time video applications typically have limited knowledge of frame sizes and require strict bounds on the delay between the server and the clients. Due to this restriction, existing real-time smoothing methods [1,13,16,23] depend on approximation techniques which try to predict the relationship between the current transmission rate and the future display patterns. Still, techniques for transmitting real-time video typically cannot capitalize on a large client buffer to smooth the stream on a large time scale. Thus, real-time video streams can still exhibit significant burstiness.

An important set of emerging multimedia applications, such as live video-casts of course lectures or television news, lie between the two extremes of stored and real-time video. In these applications, many clients may be willing to tolerate a playback delay of several seconds or minutes in exchange for a smaller bandwidth requirement. This is especially true for receivers with low-rate connections to the network. Bandwidth smoothing for these live video applications is referred to as *online smoothing*.

Rexford et al. [19] proposed the first online smoothing model. In their model, stored video smoothing is used to smooth small segments of the live video as frames are generated; that is, the online algorithm is executed repeatedly during the video transmission, each time on different time slots. Similar to stored video smoothing, online smoothing tries to find the optimal transmission rate, which is bounded by two constraints: the lower bound and the upper bound. The lower bound is used to ensure continuous playback at the client site. Different from the stored video smoothing, the upper bound in online smoothing (the amount of prefetching) is limited by the video data available at the server and the buffer space available at the client, whichever is smaller. Since complete a priori information is not known, online smoothing is different from stored video smoothing. Because live applications can tolerate long delay, online smoothing can utilize a larger client buffer to smooth the video stream, which makes it different from real-time video smoothing.

In this paper, we study the problem of online VBR video traffic smoothing. In order to measure the effectiveness of online video smoothing methods, we propose a benchmark algorithm, which provides an upper bound on some of the performance metrics. Based on this algorithm, we found that significant discrepancy exists between the results produced by the existing online smoothing algorithms and the upper bound. With this observation, we focus on designing algorithms that can improve the smoothing results. Experimental results show that our algorithms make considerable improvements in some of the performance metrics compared to the existing smoothing algorithm.

The rest of the paper is organized as follows. Section 2 presents preliminaries for stored video smoothing. In Section 3, we define an online smoothing model and present a benchmark algorithm under such a model. A family of online smoothing algorithms are proposed and evaluated in Section 4. In Section 5, detailed experiments are carried out to evaluate the performance of the proposed online smoothing algorithms. Section 6 concludes the paper.

2. Stored video smoothing

A multimedia server can substantially reduce the rate requirements for transmitting stored video by prefetching frames into the client playback buffer in advance of each burst. A class of bandwidth smoothing algorithms capitalizes on a prior knowledge of the pre-recorded stream to compute a server transmission schedule, based on the size of the playback buffer.

2.1. Smoothing constraints

Consider an N-frame video stream, where frame i is f_i bytes long, $i = 1, 2, \ldots, N$. The entire video stream is stored at the server, and is transmitted across the network to a client which has B bytes playback buffer. Without loss of generality, we assume a discrete time-model where one time unit corresponds to the time between successive frames; for a 30 frames per second full motion video, a frame time corresponds to 1/30 s. To permit continuous playback at the client site, the server must always transmit enough data to

avoid buffer underflow at the client, where

$$F_{\text{lower}}(t) = \sum_{i=1}^{t} f_i$$

indicates the amount of data consumed by the client by frame time t, where $t = 1, 2, \ldots, N-1$. In addition, a client should receive no more than

$$F_{\text{upper}}(t) = B + \sum_{i=1}^{t} f_i$$

by frame time t to prevent buffer overflow of the playback buffer (of size B). Informally, the upper bound function F_{upper} is obtained by shifting the lower bound function F_{lower} up by B. Consequently, any valid server plan should stay within the constrained region defined by F_{upper} and F_{lower}. That is:

$$F_{\text{lower}}(t) \le \sum_{i=1}^{t} c_i \le F_{\text{upper}}(t)$$

where c_i is the transmission rate during frame slot i of the smoothed video stream.

2.2. Creating transmission plans

A bandwidth smoothing plan draws a path from the beginning of the movie to the end that stays within the region defined by the functions F_{upper} and F_{lower}. The path consists of many change points, where the transmission rate is changed. The straight line between two change points is referred to as a *run*. Thus, creating a bandwidth plan involves generating m consecutive runs each with a constant bandwidth allocation r_j and a duration t_j; at frame time i, the transmission rate $c_i = r_j$, where frame slot i occurs within the jth run. Together, the m runs must form a monotonically increasing, piecewise-linear path that stays between the F_{lower} and F_{upper} curves. For example, Fig. 1 shows a plan with $m = 3$ runs, where the second run reduces the transmission rate to avoid buffer overflow at the client

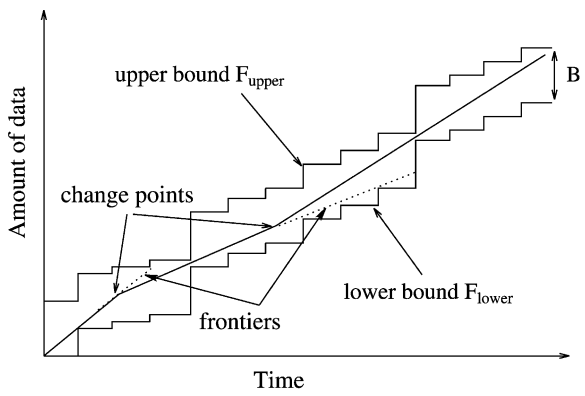


Fig. 1. Traffic smoothing for stored video.

prefetch buffer; similarly, the third run increases the transmission rate to prevent underflow.

Given a starting point for the $(j+1)$th run, most smoothing algorithms attempt to select a trajectory that extends as far as possible, to limit the number of bandwidth changes during the reminder of the plan. As a result, the trajectory for each run must eventually reach both the overflow and the underflow curves, generating a *frontier* of possible starting points for the next run, as shown in Fig. 1. Reducing the peak transmission rate of the video stream is the primary goal of each of the smoothing algorithms. The algorithms, however, differ in how they select a starting point for the $(j+1)$th run on rate increases and decreases, resulting in transmission plans with different performance properties. For example, the critical bandwidth allocation (CBA) algorithm [5] starts a rate decrease at the leftmost point on the frontier, where the trajectory for run j hits the F_{lower} curve; for rate increases, the CBA algorithm performs a search along the frontier to locate the starting point that allows the next trajectory to extend as far as possible. For any rate change, the CBA algorithm determines the longest trajectory for the $(j+1)$th run, based on the selected starting point and initial buffer occupancy (vertical distance from F_{lower}). This results in a transmission plan that has the smallest peak bandwidth requirement and the minimum number of bandwidth increases [5]. Instead of minimizing the number of rate increases, other smoothing algorithms create plans to optimize different performance metrics.

- The MCBA algorithm [6,25] minimizes the total number of rate changes.
- The MVBA algorithm [21] minimizes the variability of transmission rate.
- The RCBS [3] minimizes the utilization of the prefetch buffer.
- The ON–OFF algorithm [24] minimizes the number of on/off segments.
- The PCRTT [15] algorithm minimize cost metrics through dynamic programming.

A more in-depth discussion of these algorithms can be found in Ref. [4].

3. The benchmark algorithm for online video smoothing

Existing techniques for transmitting stored video serve as a useful foundation for developing new algorithms for online smoothing under client buffer constraints. However, operating with a limited available data and limited knowledge of future frame sizes requires important changes to the model in Fig. 1. In this section, we formulate an online smoothing model, and propose a benchmark online smoothing algorithm based on the model.

3.1. Online smoothing model

Without loss of generality, we consider a discrete time model at the granularity of a frame time. At any time t, the smoothing server has knowledge of the next W frame sizes, where W to referred to as the lookahead window. Beyond these W frames, the server can only predict future frame sizes. We assume that the buffer size at the server is large enough to buffer the largest W consecutive frames, and then the server has a flexibility to adjust the transmission rate to avoid underflow and overflow. To avoid underflow, the server should transmit enough data to allow the client to drain its buffer at the frame display rate; to avoid overflow, The server must limit the amount of prefetching to prevent overflow of this buffer, and the server cannot transmit any frame later than frame $t + W$ at time t.

We use the same function F_{lower} as that in Section 2. For the upper bound F_{upper}, however, we use the following function:

$$F_{\text{upper}}(t) = \min\{F_{\text{lower}}(t) + B, F_{\text{lower}}(t + W)\} \quad (1 \le t \le N)$$

Where B is the buffer size in bytes, W is the lookahead window (delay) in number of frames, and N is the number of frames in the video. Note that when the server receives the entire video by time N, the transmission can continue until time $N + W$, since playback at the client is delayed by W frame times to enable smoothing, Thus, we assume:

$$F_{\text{lower}}(t) = F_{\text{lower}}(N) \quad (t > N)$$
$$F_{\text{lower}}(t) = 0 \quad (t = 0)$$

Fig. 2 further explains the online smoothing model. To create a bandwidth plan, each frame i is examined and F_{upper} is calculated for that point. Figs. 1 and 2(a) show the buffer constraint and the time constraint, respectively. Informally, the buffer constraint function is obtained by shifting the lower bound function F_{lower} up by B. The time constraint function is obtained by shifting F_{lower} to the left by W. As shown in Fig. 2(b), the new upper bound F_{upper} is the minimum of the time constraint and the buffer constraint.

To generate a transmission schedule, the server could conceivably compute a new schedule (using any bandwidth smoothing algorithm) at every time unit to incorporate the most recently available frame size information. To reduce computation complexity, the server could instead execute the smoothing algorithm only when the client buffer is close to full or empty.

3.2. The benchmark algorithm

Based on F_{lower} and F_{upper}, we can use one of the off-line smoothing algorithm to create a plan that represents 'the best we could have done'. The creation of such a plan is similar to post processing page replacement algorithms in the operating system domain to figure out what performance was actually achievable. In practice, this optimal plan is unobtainable since the server uses the knowledge of the frame sizes of the whole video, but the server only has knowledge of the next W frame sizes at any time. However, such an off-line smoothing algorithm provides an upper bound on the smoothing achievable given the time and buffer constraints, which is useful for performance comparison. The pseudo-code for the benchmark algorithms is shown in Fig. 3.

Different bandwidth smoothing algorithms may optimize different performance metrics. The appropriate optimization criterion depends on the underlying resource allocation model at the server and client sites, as well as the network. In a realistic setting, however, more than one criterion may have impact on the resource requirements, particularly when the components in the system have different performance goals. For example, low buffer utilization may appeal to the clients, whereas low bandwidth variability may appeal to service providers that wish to multiplex as many streams as possible. As a result, it becomes important to understand how well each smoothing algorithm performs across the range of possible optimization criteria. An algorithm that has

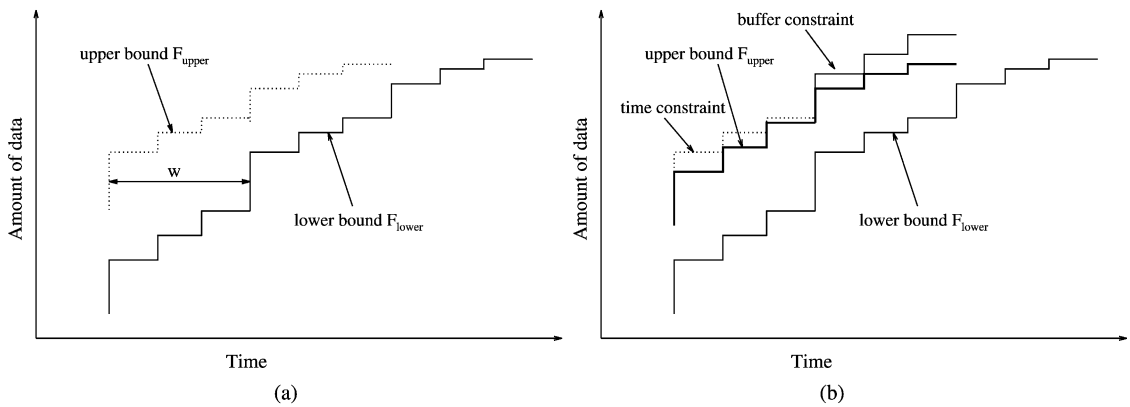


Fig. 2. Upper bound and lower bound curves for online smoothing.

for (i=1; i < N; i++)
$F_{lower}[i] = F_{lower}[i-1] + f_i$;
for (i=1; i < N; i++)
if ($F_{lower}[i] + B < F_{lower}[i+W]$)
$F_{upper}[i] = F_{lower}[i] + B$;
else $F_{upper}[i] = F_{lower}[i+W]$;
Run a bandwidth smoothing algorithm using F_{upper} and F_{lower} to get the bandwidth plan.

Fig. 3. The benchmark algorithm.

near-optimal performance according to several metrics may be preferable to an algorithm that is optimal for one metric and performs poorly for all the others. Based on the experimental results in Ref. [4], the CBA algorithm has competitive performance with other algorithms in terms of peak bandwidth, coefficient of variation and the number of rate changes. Thus, in this paper, we use the CBA algorithm [5] as the bandwidth smoothing algorithm in our benchmark algorithm (certainly, other algorithms can also be used).

4. Heuristics for online video smoothing

In this section, we first evaluate the SLWIN algorithm [19], which is the first online smoothing algorithm. Then, we propose a family of online smoothing algorithms: the base algorithm, the predict algorithm, and the hold algorithm, to address the weakness of the SLWIN algorithm.

4.1. The SLWIN algorithm

In Fig. 4, at time t, the server does not have any information about frames later than the $t + W$ frame. Thus, a transmission plan should be based on the existing video frames. Since a transmission rate should not be greater than r_{max} to avoid overflow, and should not be less than r_{min} to avoid underflow, a transmission rate should be bounded by

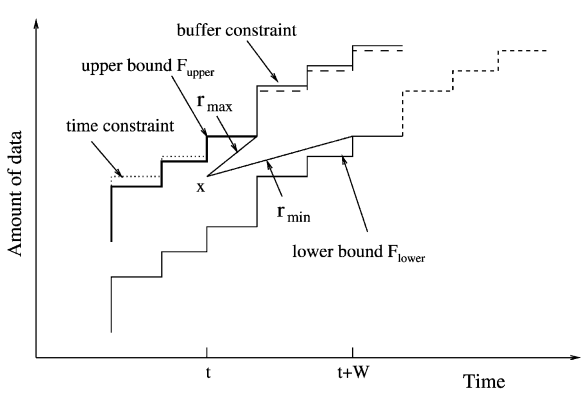


Fig. 4. The SLWIN algorithm.

r_{max} and r_{min}. The SLWIN algorithm always uses r_{min} as the new transmission rate. Since any transmission rate between r_{min} and r_{max} can be used, we consider three transmission plans:

- The *min* approach which uses r_{min} as the transmission rate.
- The *max* approach which uses r_{max} as the transmission rate.
- The *ave* approach which uses $r_{ave} = (r_{min} + t_{max})/2$ as the transmission rate.

We evaluate these transmission plans by using a video trace from the MPEG movie *Star Wars* [9]. Fig. 5 shows the smoothing results with $B = 1$ MB and $W = 600$ frames. As can be seen, the *min* approach has smaller peak rates and burstiness compared to the *ave* approach, while the *ave* approaches exhibits smaller peak rate and burstiness than the max approach.

Since the *min* approach always uses r_{min} as the transmission rate, its buffer occupancy is pretty low. For example, on the right graph of Fig. 5, the client buffer occupancy rate in the min approach is always less than 20%. For the same reason, the *max* approach keeps the client buffer occupancy close to 100%, and the *ave* approach keeps the buffer occupancy around 50%.

When the server transmission rate is larger than the client stream playback rate, the client buffer builds up; when the transmission rate is less than the client stream playback rate, the client buffer shrinks. In order to keep the same transmission rate for a long time (i.e. to smooth the video frames), the client buffer should be fully utilized. The *min* approach, the *ave* approach, and the *max* approach only use a small fraction of the client buffer, and hence these approaches cannot effectively smooth out the burstiness. Based on this observation, we propose an online smoothing algorithm, called *base* algorithm, to effectively utilize the client buffer, and then can effectively smooth the video stream.

4.2. The base algorithm

Fig. 6 shows the pseudo-code of the base algorithm. In the base algorithm, the new transmission plan is constrained by r_{min} and r_{max}. If the previous transmission rate is inside the region defined by r_{min} and r_{max}, the transmission rate will not be changed. If the previous transmission rate is larger than r_{max}, to avoid overflow, the transmission rate should be reduced. In the base algorithm, the new transmission rate uses r_{max} to minimize the variability of transmission rate. If the previous transmission rate is less than r_{min}, r_{min} will be used as the new transmission plan.

The base algorithm is different from the *min* approach, the *ave* approach, and the *max* approach, since it can effectively utilize the client buffer. The *min* approach uses r_{min} as the transmission rate, and then the client buffer is

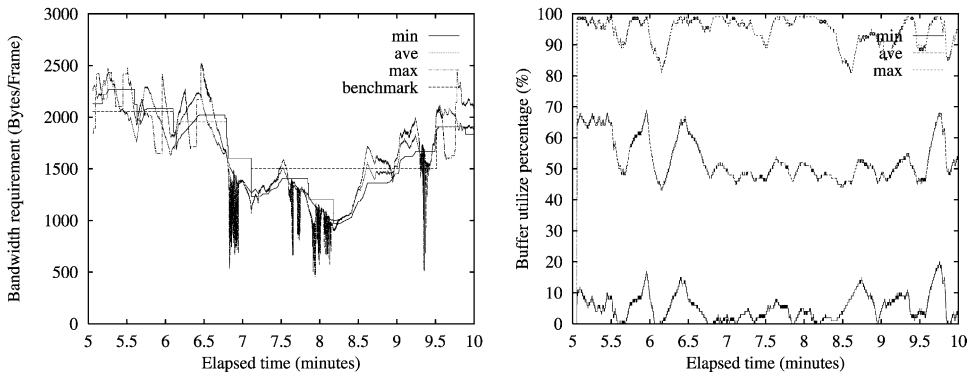


Fig. 5. Comparison of three transmission plans: *min*, *ave*, and *max* with $B = 1$ MB; $W = 600$ frames. The graphs on the left plot the bandwidth requirements of different transmission plans. The graphs on the right plot the client buffer utilization percentage of different transmission plans.

always near empty. The *ave* approach uses r_{ave} as the transmission rate, and then the client buffer is always near 50%. The max approach uses r_{max} as the transmission rate, and then the client buffer is always near full. In the base approach, the transmission rate changes between r_{min} and r_{max}. When the transmission rate is close to r_{max}, the client buffer is more likely to grow and even near full. Later, if there are many complex scene changes, the frame size may be increased. By keeping the original transmission rate, the client drains its buffer to keep the frame display rate. After some time, the original transmission rate may be close to the new r_{min}, and the client buffer shrinks, and even close to empty.

Fig. 7 demonstrates the advantage of our base algorithm: the results are much smoother than the *min* approach. (Recall that the *min* approach is smoother than the *ave* approach and the *max* approach.) This can be explained by looking at the buffer utilization graph. As can be seen, the buffer utilization varies from near 0 to 97% in the base algorithm, but in the *min* approach, the buffer utilization is never greater than 20%. From time 6 to 6.3, there is a burst in the *min* algorithm. The burst is smoothed out by the base algorithm since the client can maintain the video display rate by draining the buffered data instead of increasing the transmission rate. At time 6, the client buffer utilization in the base algorithm reaches about 60%, which can never be achieved in the *min* approach.

Although the base algorithm performs better than the *min* approach, the *ave* approach, and the *max* approach, it is still burstier than the benchmark algorithm. For example, on the left graph of Fig. 7, the transmission rate of the base algorithm fluctuates several time from time 7.2 to time 9.7, where the bench-mark algorithm keeps the same transmission rate. Next, we propose novel techniques to further improve the performance.

4.3. The predict algorithm

Based on the online smoothing model, at any time t, the smoothing server only has knowledge of the next W frames of data. Beyond these W frames, the server can only predict future frame size. The benchmark algorithm has better performance than the base and the hold algorithms since it has the knowledge of future frames sizes. Based on this observation, correctly predicting future frame size should be able to improve performance. As shown in Fig. 8, if future information is known, r'_{min} and r'_{max} should be r_{min} and r_{max}, respectively. Then, the same transmission rate can last longer, and get a smoother result. In the following, we propose techniques to predict future frame sizes.

Predict future frame sizes: In MPEG video, there are three types of encoded pictures: I (intra-coded), P (predicted), and B (bidirectional). Usually, an I frame is 2–5 times larger than a P frame, which is 2–5 times larger than a B frame. The sequence of encoded pictures is specified by two parameters: M, the distance between I or P pictures, and N, the distance between I pictures. Thus, if M is 3 and N is 9, the sequence of encoded pictures is:

I B B P B B P B B I B B P B B...

where the group of picture (GOP) pattern IBBPBBPBB repeats until the end of the movie.

There are two reasons for the burstiness of MPEG video. First is due to the IPB picture change from one frame to the other. The other is due to scene changes; that is, pictures of more complex scenes require more bits to encode. We observed that the output rates from one scene to the next differ by about a factor of 3 in the worst case, which is relatively small compared to the IPB picture change. Thus,

Based on the available frame sizes, create F_{lower} and F_{upper}, and then calculate r_{max} and r_{min}.
if $(last\_bw > r_{max})$
    $rnew = r_{max}$;
else if $(last\_bw < r_{min})$
    $rnew = r_{min}$;
else $rnew = last\_bw$;
return $rnew$.

Fig. 6. The base algorithm.

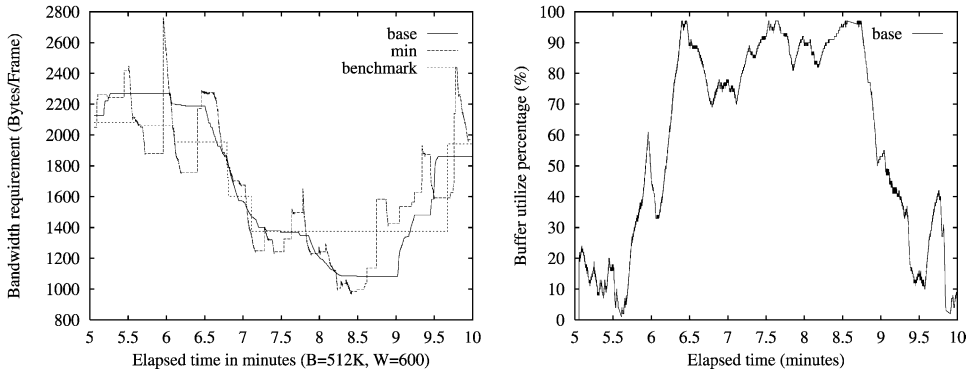


Fig. 7. Comparison of the *min* approach, the base algorithm, and the benchmark algorithm. The graphs on the left plot the bandwidth requirements of different transmission plans. The graphs on the right plot the client buffer utilization percentage of different transmission plans.

we can get a pretty good approximation if we use the IPB pattern to predict future frame sizes. For an MPEG video with a GOP size of N_g, the estimated size for frame i is simply chosen to be the size of frame $i - N_g$. This approximation is similar to the approach proposed by Lam et al. [13].

By using the GOP pattern, we can predict future frame sizes, and then we get a more constrained upper bound $F_{\text{upper}} = \min i(F_{\text{time}}, F_{\text{buffer}})$. Inside the region defined by F_{upper} and F_{lower}, we run the base algorithm to find a transmission plan. Whenever there is a need to change the transmission rate, the base algorithm either chooses $r_{\min}$ or $r_{\max}$. However, there are other options between $r_{\min}$ and $r_{\max}$. Next, we propose a predict algorithm that can explore these opportunities.

The predict algorithm: We have the following observations:

Observation 1: If the average size of the last J frames is larger than the average size of the last K ($K \gg J$) frames, the video stream has an increase tendency, and then a transmission rate larger than $r_{\min}$ should be used.

Observation 2: If the average size of the last J frames is smaller than the average size of the last K ($K \gg J$) frames, the video stream has a decrease tendency, and then a transmission rate smaller than $r_{\max}$ is better.

The predict algorithm modifies the base algorithm by considering the above two observations. Fig. 9 shows the pseudo-code of the predict algorithm, where α, α', β, β', J, and K are system tuning factors. α' controls when to apply the prediction. By choosing a large α' the chances of applying the prediction will be increased, and then $\alpha \times r_{\max}$ is more likely to be used. α decides how much percent the new transmission rate should be reduced compared to $r_{\max}$. The definitions of β and β' are similar to the definitions of α and α'. Typically, we choose $\beta = -2 - \alpha$, $\beta' = 2 - \alpha'$. Note that $\alpha \times r_{\max} > r_{\min}$, and $\beta \times r_{\min} < r_{\max}$, since the new transmission rate should be bounded by $r_{\min}$ and $r_{\max}$. Due to the use of the GOP pattern, the constraint region may be much smaller than the region used in the base algorithm (in Fig. 7).

We choose two different parameter values for the predict algorithm and refer them as the 90% approach and the 95% approach. In the 90% approach, $\alpha = \alpha' = 0.9$, $\beta = \beta' = 1.1$. In the 95% approach, $\alpha = \alpha' = 0.95$, $\beta = \beta' = 1.05$. Fig. 10 compares the performance of these two predict algorithms with the benchmark algorithm and the base

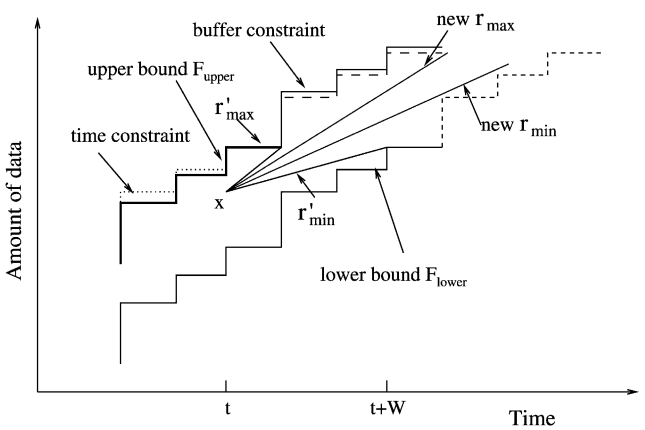


Fig. 8. Determination of $r_{\min}$ and $r_{\max}$ with knowledge of future frame sizes.

1: Based on the available and predicted frame sizes, create F_{lower} and F_{upper}, and then calculate r_{max} and r_{min}.
2: **if** $(last\_bw > r_{max})$
3: **if** $(\frac{1}{J} * \sum_{i=t+W-J}^{t+W} f_i < \alpha' * \frac{1}{K} * \sum_{i=t+W-K}^{t+W} f_i)$
4: $rnew = \alpha * r_{max}$;
5: **else** $rnew = r_{max}$;
6: **else if** $(last\_bw < r_{min})$
7: **if** $(\frac{1}{J} * \sum_{i=t+W-J}^{t+W} f_i > \beta' * \frac{1}{K} * \sum_{i=t+W-K}^{t+W} f_i)$
8: $rnew = \beta * r_{min}$;
9: **else** $rnew = r_{min}$;
10: **else** $rnew = last\_bw$;
11: **return** $rnew$.
where $0 < \alpha, \alpha' < 1$, $\beta, \beta' > 1$, $J << K$, $\alpha * r_{max} > r_{min}$, and $\beta * r_{min} < r_{max}$.

Fig. 9. The predict algorithm.

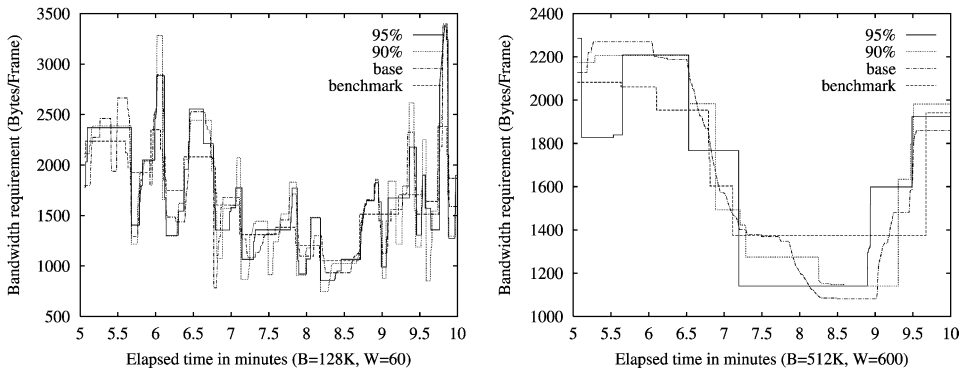


Fig. 10. Comparison of the benchmark algorithm, the base algorithm, the 90% and the 95% approach.

algorithm. The results show that the 95% approach is smoother than the 90% approach and the base algorithm, but there are still some discrepancies compared to the benchmark algorithm. For example, on the right graph of Fig. 10, with 20 s lookahead window ($W = 600$ frames), the benchmark algorithm keeps the same transmission rate from time 7.2 to 9.7, but the transmission rate of the predict algorithms changes several times.

4.4. The hold algorithm

In the base algorithm and the predict algorithm, whenever a frame is generated, a new transmission plan is created to incorporate the most recently available frame size information. This may increase the computation overhead substantially. To reduce the computation overhead, we propose a new algorithm which is referred to as the *hold* algorithm.

We observe that there is no need to change the transmission rate if the buffer is not near underflow or overflow. Thus, in the hold algorithm, we try to keep the original transmission rate as long as possible. Fig. 11 shows the pseudo-code of the hold algorithm, where D_t represents the amount of data in the client buffer. α and β are system tuning parameters, which are used to control how long the current transmission rate should be hold.

The hold algorithm can reduce the number of rate changes and the computation overhead substantially. However, the hold algorithm has a poor performance in terms of peak rate and variability of transmission rate (see Section 5), since it is not quite active; e.g. if there is a big scene change, the transmission rate should be changed earlier even though the buffer is not near underflow and overflow; otherwise, a big bandwidth increase or decrease may be necessary.

Let $\mu = \frac{D_t - F_{lower}}{F_{upper} - F_{lower}}$;
if $(\alpha < \mu < \beta)$
    keep the original bandwidth;
else call the base algorithm.
where $0 < \alpha < \beta < 1$

Fig. 11. The hold algorithm.

5. Performance evaluation

We studied the performance of the SLWIN(1) algorithm,[1] the base algorithm, the hold algorithm, and the predict algorithm through extensive experiments. The experiments are conducted using a video trace from the MPEG movie *Star Wars* [9], which has a mean frame size of 1950 bytes, a viewable size of 320×240, and a frame rate of 30 frames per second. By changing the lookahead window size W, the client buffer size B, and the prediction parameters, different online smoothing algorithms are evaluated by three performance metrics: the *peak rate*, the *coefficient of variation* (COV), and the number of *rate changes*.

5.1. Performance metrics

Peak rate: The peak rate of a smoothed video stream determines the worst-case bandwidth requirement across the path from the video storage on the server, the route through the network, and the playback buffer at the client site. Hence, most bandwidth smoothing algorithms attempt to minimize $\max\{r_j\}$ (r_j is the transmission rate of the jth run) to increase the likelihood that the server, network, and the client have sufficient resources to handle the stream. This is especially important if the service must reserve network bandwidth based on the peak rate, or if the client has a low-bandwidth connection to the network. In addition, reducing the peak rate permits the server and the network to provide deterministic guarantees to a larger number of streams.

[1] The SLWIN(1) algorithm [19] is the same as the min approach. Since it outperforms other SLWIN algorithms [19], we only compare the SLWIN(1) algorithm with the proposed online smoothing algorithms.

Variability of transmission rate: In addition to minimizing the peak rate, a smoothing algorithm should reduce the overall variability in the rate requirements for the video stream. Intuitively, plans with smaller rate variation should require fewer resources from the server and the network; more precisely, smoother plans have lower effective bandwidth requirements, allowing the server and the network to statistically multiplex the maximum number of streams [26]. Even under a deterministic model of resource reservation, the server's ability to change a stream's bandwidth reservation may depend on the size of the adjustment ($|r_j + 1 - r_j|$), particularly on rate increases. If the system does not support advance booking of resources, the server or the network may be unable to acquire enough bandwidth to start transmitting frames at higher rate. We use coefficient of variation (COV)

$$\frac{\text{stdev}\{c_0, c_1, \ldots, c_{n-1}\}}{\frac{1}{n}\sum_{i=0}^{n-1} c_i}$$

to normalize the variability metric across different streams, where the server transmits at rate c_i at time i.

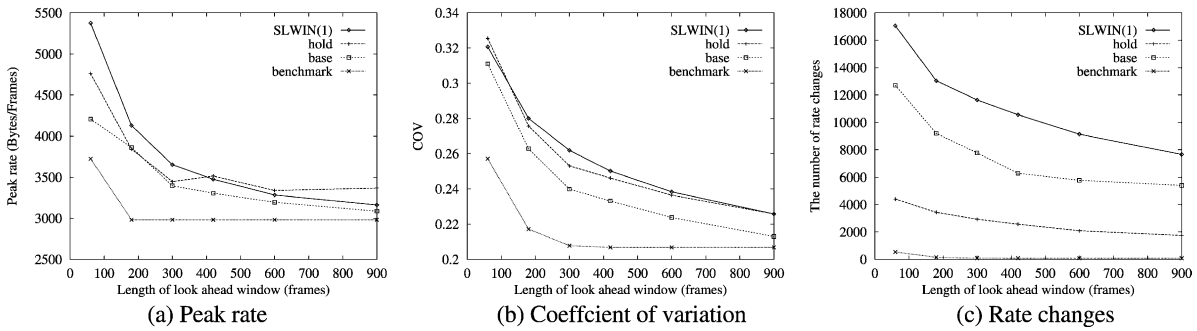
Number of rate changes: In addition to reducing the variability in resource requirements, bandwidth smoothing algorithms should also decrease the frequency of rate changes. It is especially critical for live video-casts where network resource requirements cannot be determined at connection set-up time. Thus resources are either reserved statically based on a conservative estimate or dynamically renegotiated according to the changes in the smoothing result. In many cases, the dynamic approach is a preferable choice for resource reservation since it may substantially improve resource utilization and reduce connection cost. Under the dynamic approach, decreasing the number of rate changes reduces the cost (or number) of negotiating with the network [10]. It is therefore essential to keep the number of rate changes as small as possible.

5.2. Lookahead window size (W)

Figs. 12 and 13 compares the performance of the SLWIN(1) algorithm [19], the base algorithm, the hold algorithm, and the benchmark algorithm across a range of lookahead window sizes when the buffer size is 512 KB and 1024 KB, respectively.

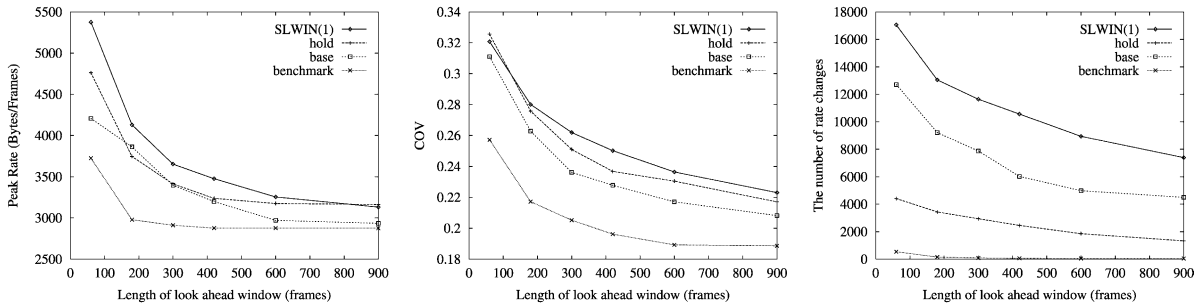
It is obvious that the benchmark algorithm has the best performance in terms of peak rate, COV, and the number of rate change. Having complete knowledge of future frame sizes enables the server to smooth future bursts by aggressive workahead transmission. However, since the server does not have access to the actual future video frames, workahead at any instant is limited by the amount of data present at the server. This explains the fact that the benchmark algorithm reduces the peak rate by about 20% when the lookahead window increases from 2 s (60 frames) to 6 s (180 frames).

Increasing the lookahead window size increases the startup delay at the client. This has two important implications for smoothing. A larger window gives the server more time to transmit any frame; a larger window also allows the server to perform workahead transmission more aggressively, as it has access to a larger number of frames. Because of this, for a given client buffer size, as the size of the smoothing window increases, the peak rate, COV, and the number of rate changes decrease at first. For example, in Fig. 12 (and Fig. 13), the benchmark algorithm, the hold algorithm, and the SLWIN(1) algorithm reduce the peak rate by about 20% when the lookahead window increases from 2 to 6 s. However, increasing W beyond a certain point does not help as the client buffer becomes the limiting constraint on smoothing. For example, in Fig. 12, with $B = 512$ KB, when the window size is larger than 300 frames, since $300 \times 1950 > 512$ KB, the upper bound function is decided by the buffer size instead of the time constraint. Thus, the peak rate, the COV, and the number of rate changes of the benchmark algorithm do not change when the window sizes are larger than 300 frames (The curves flat out in this region). Similarly, in Fig. 13, with $B = 1$ MB, the bandwidth metrics do not change when the window size is larger than 600 frames in the benchmark algorithm. This is not exactly true for other algorithms. For example, in Fig. 13, the peak rate decrease trend of the base algorithm slows down when the lookahead window size passes 20 s (600 frames), but it still reduces the peak rate by



(a) Peak rate (b) Coeffcient of variation (c) Rate changes

Fig. 12. Comparison of algorithms under different window sizes ($B = 512$ KB). (a) Peak rate. (b) Coefficient of variation. (c) Rate changes.



(a) Peak rate (b) Coeffcient of variation (c) Rate changes

Fig. 13. Comparison of algorithms under different window sizes ($B = 1$ MB).

about 1% when the lookahead window size increases from 20 to 30 s. This can be explained as follows. Although the upper bound is constrained by the buffer size when $W > 600$, knowing future frame sizes helps choose a transmission rate which can last longer. This is different from the benchmark algorithm, where the whole video frame sizes are known, and then it is not helpful to increase the lookahead window when the client buffer is the constraint.

From Figs. 12 and 13, we can see that the base algorithm consistently outperforms the SLWIN(1) algorithm. For example, in Fig. 13, the base algorithm consistently reduces the peak rate by more than 10% compared to the SLWIN(1) algorithm. As we explained in Section 4.2, the base algorithm can effectively utilize the client buffer, but the SLWIN(1) (same as the min approach) always keeps the buffer utilization low.

The hold algorithm fluctuates when compared to the SLWIN(1) algorithm in terms of COV and peak rate, but it performs worse than the base algorithm. This suggests that the higher variability in the hold algorithms stems from its attempts to combine multiple bandwidth changes into a single transmission rate, rather than making gradual adjustments. Since the hold algorithm is not quite active, if there is a big scene change, the transmission rate has to make a big increase or decrease, which increases the peak rate and COV. However, holding the transmission rate has some advantage in terms of rate changes. In Fig. 12, with 2 s lookahead window, the hold algorithm cuts the rate changes to a factor of 3 compared to the base algorithm, and cuts the rate changes to a factor of 4 compared to the SLWIN algorithm.

5.3. Buffer size (B)

For a small lookahead window, the main constraint on smoothing is the window size and then the performance is similar across different client buffers. For example, in Figs. 12 and 13, when the window size is smaller than 300 frames, the peak rate, the COV, and the number of rate changes do not change when the size of the client buffer changes from 512 KB to 1 MB. As the window size increases, the advantage of a larger client buffer becomes clear. For example, with a 20 s (600 frame) window, increasing the client buffer size from 512 KB to 1 MB reduces the peak rate by almost 10%.

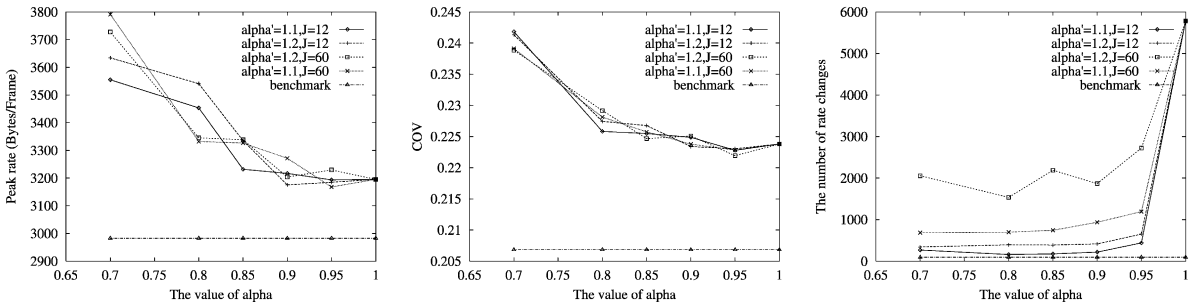
For small lookahead window sizes, the client buffer is still large enough that it does not constrain the allowable workahead. For larger windows, however, the client buffer limits the amount of workahead; the server has enough data but is constrained by the relatively small client buffer. In addition, for such buffer size, large smoothing windows allow the online smoothing algorithm to achieve peak rates which are close to the benchmark schedule. For example, with a 1 MB client buffer and a 30 s lookahead window, the peak rate of the smoothed schedule in the base algorithm is only 2% larger than the peak rate of the benchmark algorithm.

Since the SLWIN algorithm can only make use of a small percentage of the client buffer, the peak rate, the COV, and the number of rate changes do not decrease as much as other algorithms when the client buffer size increases. For example, with $W = 600$ frames, in the base algorithm, the COV decreases more than 3% when the buffer size increases from 512 KB to 1 MB; in the SLWIN(1) approach, the COV only decreases less than 1%.

5.4. The effectiveness of prediction

The evaluated algorithms, except the benchmark algorithm, assume that the lookahead information consists of the actual future frame sizes; i.e. the server can only have knowledge of the next W frame sizes at any time. Since the benchmark algorithm uses the actual future frame sizes, and any prediction scheme cannot obtain the exact future frame sizes, the performance of the benchmark algorithm sets up an upper bound for any prediction schemes.

The effectiveness of the predict algorithm is evaluated by changing the parameters used in Fig. 9. To simplify the experiment, we assume $K = 600$, $\beta = 2 - \alpha$, $\beta' = 2 - \alpha'$. By changing the value of J and α', several predict algorithms are obtained and evaluated. The results are shown in Fig. 14. When $\alpha = 1$, $\beta = 2 - \alpha = 1$,



(a) Peak rate (b) Coeffcient of variation (c) Rate changes

Fig. 14. Comparisons of the predict algorithms ($B = 512$ KB, $W = 600$ frames).

the prediction algorithm reduces to the base algorithm. As can be seen, the performance of the predict algorithms is not stable, i.e. it varies significantly when the algorithm parameters such as α, α', J, change. Generally speaking, when α decreases, the prediction algorithm performs worse, since the transmission rate may change too much. For example, when $\alpha = 0.7$, the transmission rate may change more than 30%. This over-active may make the transmission rate fluctuate between $r_{\min}$ and $r_{\max}$, and thus degrades the performance.

The interesting fact is that the number of rate changes in the predict algorithms is significantly reduced compared to other algorithms. This can be explained as follows. When the transmission rate is larger than $r_{\max}$ and the rate has a decrease tendency, the base algorithm takes the conservative approach; e.g. use the new $r_{\max}$ for the next frame, and then it is more likely to reach the upper bound and have to use another new $r_{\max}$. The predict algorithms takes an aggressive approach; e.g. uses $\alpha \times r_{\max} (0 < \alpha < 1)$ as the transmission rate for the next frame, and then it is less likely to reach the upper bound and change the transmission rate. Thus, the number of rate changes is significantly reduced in the predict approaches compared to the base approach.

The Modified Predict Algorithm: Based on the results in Fig. 14, it is very difficult to find the optimum parameter values of the predict algorithm. We also observe that the value of these parameters should not be fixed. For example, the value of α should depend on how much percent the video frame sizes decrease. When the video stream has a big decrease tendency, the value of α should be smaller than that when the video stream has a small decrease tendency. Based on this idea, we modify the predict algorithm by changing lines 3, 4, 7, and 8 of the predict algorithm in Fig. 9 as follows:

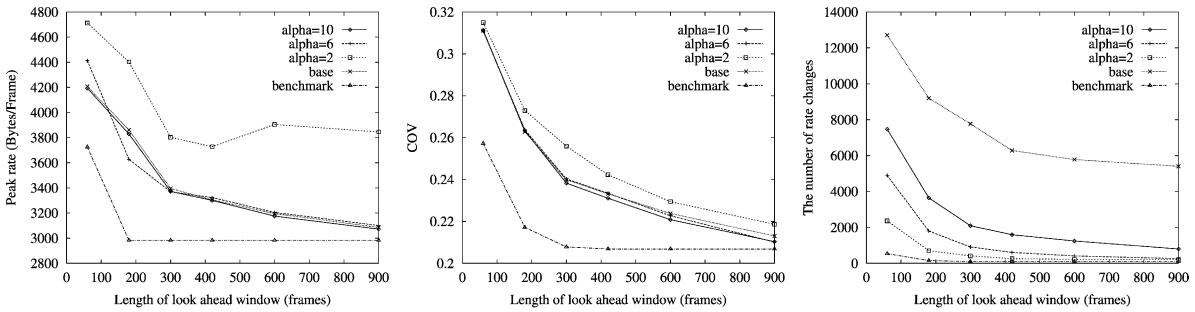
$$3: \quad \text{if} \left(I = \frac{\frac{1}{J}\sum_{i=t+W-J}^{t+W} f_i}{\frac{1}{K}\sum_{i=t+W-K}^{t+W} f_i} < 1 \right)$$

$$4: \quad \text{rnew} = \sqrt[\alpha]{I} r_{\max};$$

$$7: \quad \text{if} \left(I = \frac{\frac{1}{J}\sum_{i=t+W-J}^{t+W} f_i}{\frac{1}{K}\sum_{i=t+W-K}^{t+W} f_i} > 1 \right)$$

$$8: \quad \text{rnew} = \sqrt[\alpha]{I} r_{\min};$$

Fig. 15 evaluates the performance of the modified predict algorithm. We assume $J = 12$, $K = 600$, $B = 512$ KB. The $\alpha = 2$ approach is not stable in terms of peak rate. Other than that, the modified predict algorithm performs consistently. Moreover, the $\alpha = 10$ approach outperforms the base algorithm in terms of peak rate and COV, and significantly outperforms the base algorithm in



(a) Peak rate (b) Coeffcient of variation (c) Rate changes

Fig. 15. Comparisons of the modified predict algorithms ($B = 512$ KB).

terms of rate changes. For example, with 30 s lookahead window, the modified predict algorithm with $\alpha = 10$ cuts the rate changes to a factor of 7 compared to the base algorithm. Although the $\alpha = 10$ approach has a much better performance, discrepancy still exists between the results produced by $\alpha = 10$ approach and the benchmark algorithm. For example, In Fig. 15, with a 512 KB client buffer and a 30 second lookahead window, the peak rate of the $\alpha = 10$ approach is still 3% larger than that of the benchmark algorithm; the number of rate changes of the $\alpha = 10$ approach is almost 8 times larger than that of the benchmark algorithm. Recall that the benchmark smoothing plan is unachievable in practice since the server uses the knowledge of the frame sizes of the whole video, but the server only has knowledge of the next W frame sizes at any time. The disadvantage of the modified predict algorithm is the computation overhead, but this can be reduced by using table lookup; that is, assigning a factor for a range of numbers. We have implemented this approach and got comparable results.

6. Conclusions

In this paper, we studied the problem of online VBR video traffic smoothing, which exists between stored video traffic smoothing and real-time traffic smoothing. Since complete a priori information is not known, online smoothing is different from stored video smoothing. Because live applications can tolerate long delays, online smoothing can utilize a larger client buffer to smooth the video stream, and then it is different from real-time video smoothing. In order to measure the effectiveness of online video smoothing methods, we proposed a benchmark algorithm which provides an upper bound on some of the performance metrics in the smoothing results. Based on this algorithm, we found that significant discrepancy exists between the results produced by the existing online smoothing methods and the upper bound. With this observation, we focused on designing algorithms that can improve the smoothing results. Experimental results showed that our algorithms make considerable improvements compared to existing smoothing methods.

Although the proposed online smoothing algorithms improves the performance of existing online video smoothing algorithms, their transmission plans are still bursty compared to the benchmark algorithm. Further improvements may be possible. For example, we only tested some parameter values of the predict algorithm, but there are a large range of possible values for these parameters. Different values for these parameters can substantially affect the performance of the algorithm. Future work will concentrate on how to find a systematic way to test and find the optimal parameter values. Other approaches, such as adaptive linear prediction or neural network control theory may also be helpful.

References

[1] A. Adas, Supporting real-time VBR video using dynamic reservation based on linear prediction, Proceedings of the IEEE INFOCOM March (1996) 1476–1483.
[2] I. Dalgic, F.A. Tobagi, Performance evaluation of ATM networks carrying constant and variable-bit-rate video traffic, IEEE Journal of Selected Areas in Communications August (1997).
[3] W. Feng, Time-constrained bandwidth smoothing for interactive video-on-demand systems, Proceedings of the 13th International Conference on Computer Communication November (1997) 291–302.
[4] W. Feng, J. Rexford, A comparison of bandwidth smoothing techniques for the transmission of prerecorded compressed video, IEEE INFOCOM April (1997).
[5] W. Feng, S. Sechrest, Critical bandwidth allocation for delivery of compressed video, Computer Communications October (1995) 709–717.
[6] W. Feng, F. Jahanian, S. Sechrest, Optimal buffering for the delivery of compressed prerecorded video, Proceedings of IASTED International Conference on Networks January (1996).
[7] W. Feng, F. Jahanian, S. Sechrest, Optimal buffering for the delivery of compressed prerecorded video, ACM Multimedia Systems Journal September (1997).
[8] D. Gall, MPEG: a video compression standard for multimedia applications, Communications of the ACM April (1991) 46–58.
[9] M. Garrett, W. Willinger, Analysis, modeling and generation of self-similar VBR video traffic, Proceedings of the ACM SIGCOMM September (1994).
[10] M. Grossglauser, S. Keshav, D. Tse, RCBR: a simple and efficient service for multiple time-scale traffic, IEEE/ACM Transactions on Networking December (1997).
[11] Z. Jiang, L. Kleinrock, A general optimal video smoothing algorithm, Proceedings of the IEEE INFOCOM March (1998).
[12] T. Lakshman, A. Ortega, A. Reibman, Variable-bit-rate (VBR) video: tradeoffs and potentials, Proceedings of the IEEE May (1998).
[13] S. Lam, S. Chow, D. Yau, An algorithm for lossless smoothing of MPEG video, Proceedings of the ACMSIGCOMM August (1994) 281–293.
[14] J. McManus, K. Ross, Video on demand over ATM: constant-rate transmission and transport, Proceedings of the IEEE INFOCOM March (1996) 1357–1362.
[15] J. McManus, K. Ross, A dynamic programming methodology for managing prerecorded VBR sources in packet-switched networks, Telecommunications Systems (1998).
[16] P. Pancha, M. Zarki, Bandwidth-allocation schemes for variable-bit-rate MPEG sources in ATM networks, IEEE Transactions on Circuits and Systems for Video Technology June (1993) 190–198.
[17] E. Rathgeb, Policing of realistic VBR video traffic in an ATM network, International Journal of Digital and Analog Communication Systems October–December (1993) 213–226.
[18] A. Reibman, A. Berger, Traffic descriptors for VBR video teleconferencing over ATM networks, IEEE/ACM Transactions on Networking June (1995) 329–339.
[19] J. Rexford, S. Sen, J. Dey, W. Feng, J. Kurose, J. Stankovic, D. Towsley, Online smoothing of live, variable-bit-rate video, Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video May (1997) 249–257.
[20] O. Rose, Statistical properties of MPEG video traffic and their impact on traffic modeling in ATM systems, Proceedings of the Conference on Local Computer Networks October (1995) 397–406.

[21] J. Salehi, Z. Zhang, J. Kurose, D. Towsley, Supporting stored video: reducing rate variability and end-to-end resource requirements through optimal smoothing, IEEE/ACM Transactions on Networking August (1998) 397–410.
[22] G. Wallace, The JPEG still picture transmission standard, Communications of the ACM April (1991) 30–34.
[23] J. Zhang, J. Hui, Smoothness upper bound and approximation techniques for real-time VBR video traffic smoothing, Proceedings of the 18th IEEE Real-Time Systems Symposium December (1997).
[24] J. Zhang, J. Hui, Traffic characteristics and smoothness criteria in VBR video transmissions, Proceedings of the IEEE International Conference on Multimedia Computing and Systems June (1997).
[25] J. Zhang, J. Hui, Applying traffic smoothing techniques for quality of service control in VBR video transmissions, Computer Communications April (1998) 375–389.
[26] Z. Zhang, J. Kurose, J. Salehi, D. Towsley, Smoothing, statistical multiplexing, and call admission control for stored video, IEEE Journal on Selected Areas in Communications August (1997).