FOS: A Funnel-Based Approach for Optimal Online Traffic Smoothing of Live Video

Jeng-Wei Lin, Ray-I Chang, *Member, IEEE*, Jan-Ming Ho, *Member, IEEE*, and Feipei Lai, *Senior Member, IEEE*

***Abstract*—Traffic smoothing is an efficient means to reduce the bandwidth requirement for transmitting a variable-bit-rate video stream. Several traffic-smoothing algorithms have been presented to offline compute the transmission schedule for a prerecorded video. For live video applications, Sen *et al.* present a sliding-window algorithm, referred to as $SLWIN(k)$, to online compute the transmission schedule on the fly. $SLWIN(k)$ looks ahead W video frames to compute the transmission schedule for the next k frametimes, where $k \leq w$. Note that W is upper bounded by the initial delay of the transmission. The time complexity of $SLWIN(k)$ is $O(W * N/k)$ for an N frame live video. In this paper, we present an $O(N)$ online traffic-smoothing algorithm and two variants, denoted as FOS, $FOS1$ and $FOS2$, respectively. Note that $O(N)$ is a trivial lower bound of the time complexity of the traffic-smoothing problem. Thus, the proposed algorithm is optimal. We compare the performance of our algorithms with $SLWIN(k)$ based on several benchmark video clips. Experiment results show that $FOS2$, which adopts the aggressive workahead heuristic, further reduces the bandwidth requirement and better utilizes the client buffer for real-time interactive applications in which the initial delays are small.**

***Index Terms*—Live video, multimedia streaming, online delivery, traffic smoothing.**

I. Introduction

THERE are a growing number of video applications such as digital libraries, video archives, newscasts, and distance learning that can be accessed on various networks. For continuous video playback, the client player must render a new frame after one frametime (the period of time between two successive frames) has passed. In order to prevent the client player from starvation, the video server has to transmit the media data into the client buffer before it is going to be rendered. While the media data are retrieved from the disks or generated online, the video server can send a frame per frametime, as shown in Fig. 1(a). However, compressed video streams often exhibit significant burstiness of frame sizes (number of bits for each frame) on many time scales due to the encoding frame structure and their natural variations within and between scenes [15], [16]. This burstiness complicates the design of a multimedia system for high resource utilization, such as network bandwidth [19].

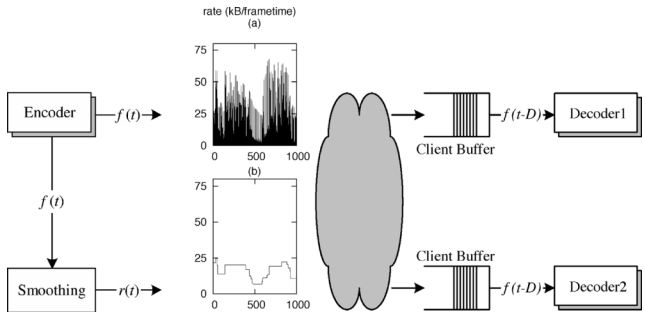


Fig. 1. Bandwidth requirement for transmitting the first 1000 frames of Star Wars is reduced from 70 to 26 kB per frametime if the output of the encoder is smoothed.

In a guarantee service model, the multimedia system has to allocate enough network bandwidth for the burstiness. However, a lot of bandwidth is wasted when the effective transmission rate is much lower than the allocated bandwidth. In a best-effort service model, when a burst suddenly jams into the network, the packets may be discarded, including those of other applications. It is necessary to design a mechanism to reduce the impact caused by the burstiness of variable-bit-rate (VBR) video streams. One approach is to smooth the burstiness. The video server can start the transmission of large frames earlier. By working ahead, it is shown that traffic smoothing is efficient at reducing the bandwidth requirement for VBR video transmission [1]–[13], as shown in Fig. 1(b).

For a prerecorded video stream, the video server has complete knowledge of all the frame sizes. The server can use a traffic-smoothing algorithm that takes advantage of the knowledge of upcoming large frames and starts more data transmission in advance of the burst. Several traffic-smoothing algorithms have been presented to offline compute the transmission schedule. For example, the Critical Bandwidth Allocation (*CBA*) algorithm [2] minimizes the total number of rate increases, the Minimum Change Bandwidth Allocation (*MCBA*) algorithm [9] minimizes the total number of rate changes, and the Minimum Variance Bandwidth Allocation (*MVBA*) algorithm [1] minimizes the variance of rate changes.

In live video applications, however, the video server only has limited knowledge of frame sizes at any time. Old media data is transmitted, while new media data is generated simultaneously. In real-time interactive applications, such as video teleconferencing, the tolerable response delay is small, so only a few frames can be buffered in the video server. In other live video applications, like newscasts and distance learning, the clients may

Manuscript received March 30, 2004; revised October 26, 2005. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Ton Kalker.

J.-W. Lin is with the Department of Information Management, Tunghai University, Taichung, Taiwan, R.O.C. (e-mail: jwlin@thu.edu.tw).

R.-I Chang is with the Department of Engineering Science and Ocean Engineering, National Taiwan University, Taipei, Taiwan, R.O.C. (e-mail: rayichang@ntu.edu.tw).

J.-M. Ho is with the Institute of Information Science, Academia Sinica, Taipei, Taiwan, R.O.C. (e-mail: hoho@iis.sinica.edu.tw).

F. Lai is with the Department of Computer Science and Information Engineering and Department of Electronic Engineering, National Taiwan University, Taipei, Taiwan, R.O.C. (e-mail: flai@ntu.edu.tw).

Digital Object Identifier 10.1109/TMM.2006.879868

be willing to tolerate a longer playback delay, of several seconds to minutes, in exchange for a smaller bandwidth requirement. For delay tolerable live video applications, Sen *et al.* introduce a sliding-window traffic-smoothing algorithm, referred to as $SLWIN(k)$, that computes the transmission schedule on the fly [10], [11]. The constant is k referred to as the slide length. Given an initial delay (D frametimes), $SLWIN(k)$ looks ahead a window of $W(k \leq W \leq D)$ frames and computes the transmission schedule for the window. After k new frames have been generated, $SLWIN(k)$ recomputes the transmission schedule. For an N frame live video, the time complexity of $SLWIN(k)$ is $O(W*N/k)$. Since the time-consuming $SLWIN(1)$ usually computes the transmission schedule of a small peak bandwidth requirement, there is a tradeoff between computing costs and performance.

In this paper, we present an $O(N)$ traffic smoothing algorithm called Funnel-Based Optimal Online Traffic Smoothing (*FOS*) for live video applications. Note that $O(N)$ is a trivial lower bound of the time complexity of the traffic-smoothing problem. Thus, the proposed algorithm is optimal. The video server can use FOS to compute the transmission schedule for a live video. In a lookahead window, FOS maintains the candidates for transmission schedules in a funnel. It considers each new frame iteratively and modifies the two chains of the funnel. If the two chains will cross each other, FOS deterministically generates a transmission schedule, or when no more frame size information is available, FOS heuristically generates a schedule at the minimal transmission rate in the funnel. While the video server executes the transmission schedule, new frames are generated, and after the transmission schedule has finished, FOS computes the next one. In fact, FOS computes the same transmission schedule as $SLWIN(1)$ in linear time.

We observe that FOS always uses the minimal transmission rate in a lookahead window. It may lower the transmission rate unnecessarily and then raise the rate in the next window. This usually occurs if the initial delay of a video transmission is small. In such cases, it is likely that all frames in the server buffer are small, e.g., B/P-frames, due to the encoding frame structure. Therefore, FOS lowers the transmission rate. When FOS finds that a large frame, e.g., I-frame, is generated later, it has to raise the transmission rate. If a traffic-smoothing algorithm has aggressively transmitted more data to the client, there are less data buffered in the server. Therefore, it is likely that the algorithm can smooth the large frame in the next window.

We have used different heuristics to improve FOS. By combining FOS with frame size prediction [13], [17], [18] and aggressive workahead [12], we derived two variant FOS algorithms called $FOS1$ and $FOS2$, respectively. Experiment results show that $FOS2$ further reduces the peak bandwidth requirement and utilizes the client buffer more efficiently when the initial delay is small.

The remainder of this paper is organized as follows. A formal definition of the online traffic-smoothing problem for live video is illustrated in Section II. Section III presents our FOS algorithm, its time complexity proof, and different heuristics. Section IV shows the experiment results. Finally, in Section V we give conclusions.

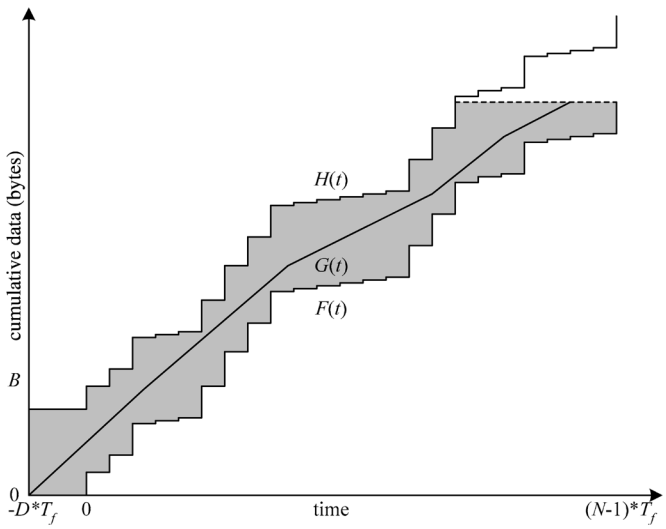


Fig. 2. Feasible transmission schedule S should satisfy $F(t) \leq G(t) \leq H(t)$, where $G(t)$ is a function of S.

II. Online Traffic Smoothing for Live Video

For an N frame video $\{f_0, f_1, f_2, \ldots, f_{N-1}; T_f\}$, which uses f_i bits to encode the ith frame and T_f is the frametime, the continuous playback schedule can be represented as the step function

$$F(t) = F_i \text{ for } i^*T_f \leq t < (i+1)^*T_f$$

where

$$F_i = \begin{cases} 0, & \text{if } i < 0 \\ F_{i-1} + f_i, & \text{if } 0 \leq i \leq N-1. \end{cases}$$

At time $i * T_f$, the client player will have played F_{i-1} bits and will continue playing f_i bits in the next frametime. Given a D frametime playback delay, a video server can transmit media data at the rate r_i from time $i * T_f$ to $(i+1) * T_f$ according to a transmission schedule $S = \{r_{-D}, r_{-D+1}, r_{-D+2}, \ldots, r_{N-2}\}$. The cumulative transmission function is defined as

$$G(t) = \begin{cases} 0, & \text{if } t \leq -D^*T_f \\ G(i^*T_f) + r_i^*(t - i^*T_f), & \text{if } i^*T_f < t \leq (i+1)^*T_f. \end{cases}$$

To guarantee continuous playback at the client, S should satisfy

$$F(t) \leq G(t) \text{ for } t \leq (N-1)^*T_f.$$

However, the client player usually does not have unlimited buffer space. We assume the client player provides a B-bit buffer. The cumulative buffer function is defined as

$$H(t) = F_{i-1} + B \text{ for } i^*T_f < t \leq (i+1)^*T_f.$$

To avoid buffer overrun, S should also satisfy

$$G(t) \leq H(t) \text{ for } t \leq (N-1)^*T_f,$$

as shown in Fig. 2.

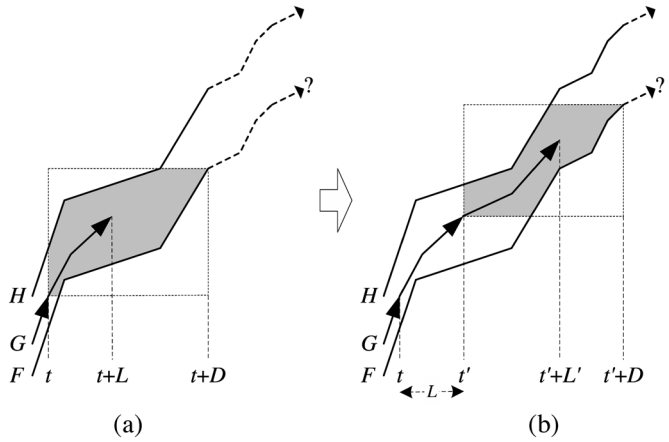


Fig. 3. Looking ahead in live video applications. (a) Online smoothing algorithm determines an L frametime transmission schedule S^t. (b) After L frametimes have passed, the algorithm determines the next transmission schedule S^{t+L}.

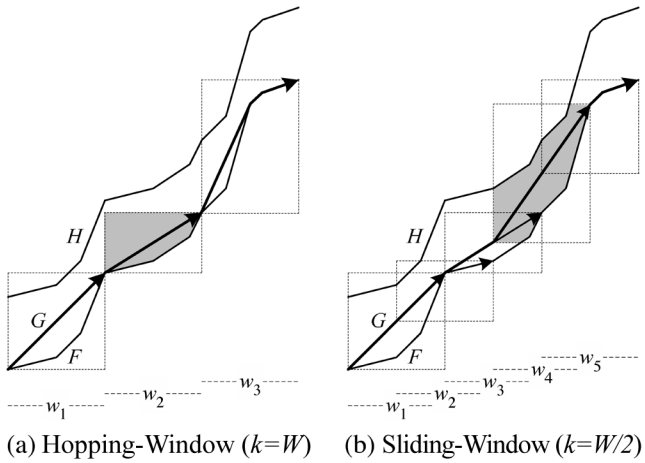


Fig. 4. (a) $SLWIN(W)$ has drawbacks when smoothing traffic across window boundaries. (b) $SLWIN(k)$ incorporates new frame size information and therefore computes a better transmission schedule.

Without loss of generality, we consider a discrete time model at the granularity of a frametime. Assuming $T_f = 1$, we simplify the definition of $F(t)$, $G(t)$ and $H(t)$ as follows.

Cumulative playback function

$$F(i) = \begin{cases} 0, & \text{if } i < 0 \\ F(i-1) + f_i, & \text{if } 0 \le i \le N-1. \end{cases} \quad (1)$$

Cumulative buffer function

$$H(i) = F(i-1) + B, \quad \text{if } i \le (N-1). \quad (2)$$

Cumulative transmission function

$$G(i) = \begin{cases} 0, & \text{if } i \le -D \\ G(i-1) + r_{i-1}, & \text{if } -D < i \le N-1. \end{cases} \quad (3)$$

As $F(N-1)$ is the total size of the video frames, a traffic-smoothing algorithm should not plan to transmit more than $F(N-1)$ data. For a prerecorded video stream, a traffic-smoothing algorithm knows each frame size and can offline compute the transmission schedule that satisfies

$$F(i) \le G(i) \le MIN\left(H(i), F(N-1)\right), \quad \text{for } -D \le i \le N-1. \quad (4)$$

For live video, however, a traffic-smoothing algorithm has limited knowledge of frame sizes at any time $t+D$ because frames after the time $t+D$ have not been generated. The algorithm $f_{t+1}, f_{t+2}, \ldots, f_{t+D}$ knows and has no idea about $f_{t+D+1}, f_{t+D+2}, \ldots, f_{N-1}$. An L $(1 \le L \le D)$ frametime transmission schedule $S^t = \{r_t, r_{t+1}, \ldots, r_{t+L-1}\}$ is feasible if S^t satisfies

$$F(i) \le G(i) \le MIN\left(H(i), F(t+D)\right), \quad \text{for } t < i \le t+L \quad (5)$$

as shown in Fig. 3(a). While the video server executes S^t, new frames are generated. After L frametimes have passed and S^t has therefore finished, L new frames have been generated. The traffic-smoothing algorithm can incorporate new frame size information to compute the next transmission schedule S^{t+L}, as shown in Fig. 3(b).

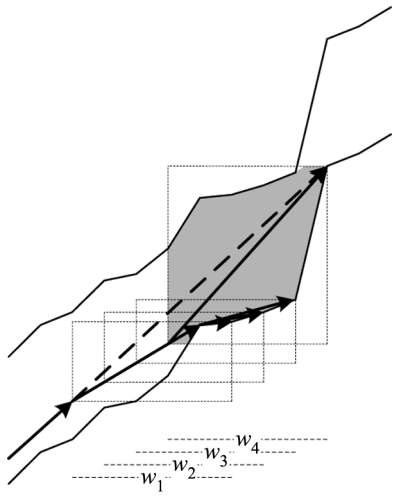


Fig. 5. $SLWIN(k)$ unnecessarily lowers the transmission rate in the first three windows and raises the rate in the fourth window.

Given a D frametime initial delay and a slide length k, $SLWIN(k)$ [10], [11] looks ahead a window of W $(k \le W \le D)$ frames and uses an $O(W)$ algorithm ($MVBA$ [1]) to compute the transmission schedule for the window. Note that $SLWIN(k)$ fixes $L = k$. After k frametimes have passed, $SLWIN(k)$ computes the next transmission schedule. The total time complexity is N/k times the complexity of smoothing a W-frame window of the video stream. Thus, the time complexity of $SLWIN(k)$ is $O(W * N/k)$.

$SLWIN(k)$ usually has a tradeoff in selecting the slide length. If the maximum slide length is used $(k = W = D)$, as shown in Fig. 4(a), the entire live video transmission is scheduled into continuous nonoverlapped transmission schedules. $SLWIN(W)$ has drawbacks when smoothing traffic across window boundaries. If k is smaller than W, as shown

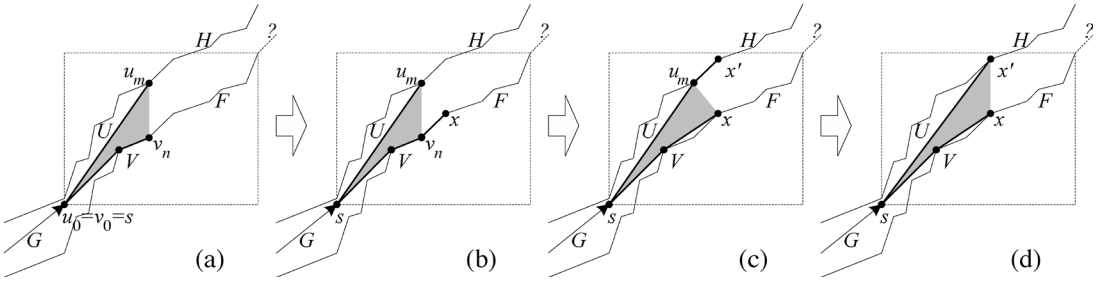


Fig. 6. FOS iteratively considers $F(a)$ and $H(a)$ for $t+1 \le a \le t+D$ to maintain the candidates for transmission schedules in the funnel, i.e., the shadowed area.

in Fig. 4(b), only the first k frametime part of the transmission schedule is used and the rest of the schedule is discarded. Although using a smaller slide length requires more computing resources, it makes a better transmission schedule.

We observe that $SLWIN(k)$ always generates the transmission schedule for a window at the minimal rate. Note that at time $t+D$, the transmission schedule that $SLWIN(k)$ generates for the window is the shortest path from $(t, G(t))$ to $(t+D, F(t+D))$ [11], [14]. $SLWIN(k)$ may lower the transmission rate unnecessarily and raise the rate in the next window, as shown in Fig. 5. This usually occurs when $SLWIN(k)$ has smoothed a window of small frames and a large frame is generated in the next window. If a traffic-smoothing algorithm has aggressively transmitted more data to the client in the previous windows, there are less data buffered in the server. Thus, it is likely the algorithm can smooth the large frame in the next window.

III. Our Algorithms and Time Complexity Analysis

A. FOS Algorithm

Like $SLWIN(k)$, at time $t+D$, our FOS algorithm tries to find the shortest path from $(t, G(t))$ to $(t+D, F(t+D))$ in the window. However, FOS generates transmission schedules in *nonoverlapped* and *variant-length* manners. It considers each frame only once and remembers useful computational results for the next window by maintaining a funnel.

As shown in Fig. 6(a), from the starting point $s = (t, G(t))$, FOS maintains the candidates for transmission schedules within a convex upper chain $U = \{u_0, u_1, u_2, \ldots, u_m\}$ and a concave lower chain $V = \{v_0, v_1, v_2, \ldots, v_n\}$, where $u_0 = v_0 = s$, $u_1, u_2, \ldots, u_m$ are points selected from the function H, and $v_1, v_2, \ldots, v_n$ are points selected from the function F. U is convex if and only if

$$m < 2 \text{ or } \mathrm{Rate}(u_0, u_1) < \mathrm{Rate}(u_1, u_2) < \ldots < \mathrm{Rate}(u_{m-1}, u_m) \quad (6)$$

where $\mathrm{Rate}(x, y)$ is the slope of the line from x to y. V is concave if and only if

$$n < 2 \text{ or } \mathrm{Rate}(v_0, v_1) > \mathrm{Rate}(v_1, v_2) > \ldots > \mathrm{Rate}(v_{n-1}, v_n). \quad (7)$$

The two chains form a funnel.

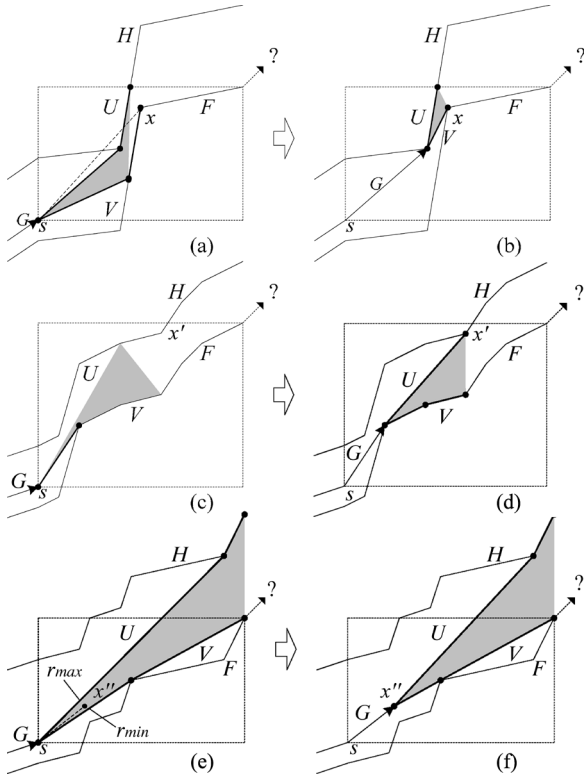


Fig. 7. (a) FOS removes $v_1, v_2, \ldots, v_n$, appends x onto the funnel, and discovers that the resultant lower chain will cross the upper chain. (b) FOS deterministically generates the transmission schedule and reconstructs the funnel. (c) FOS removes $u_1, u_2, \ldots, u_m$, appends x' onto the funnel, and discovers that the resultant upper chain will cross the lower chain. (d) FOS deterministically generates the transmission schedule and reconstructs the funnel. (e) When there is no more frame size information, FOS heuristically chooses a point x'' in the funnel. (f) FOS generates the transmission schedule and reconstructs the funnel.

We now describe how the chain points are selected. Initially, $V = \{(-D, 0)\}$, $U = \{(-D, 0)\}$, and $r_{-D} = F(0)/D$. In a window, FOS iteratively considers each unprocessed frame and modifies V and U. For $t+1 \le a \le t+D$, FOS appends the points $x = (a, F(a))$ onto V and $x' = (a, H(a))$ onto U, as shown in Fig. 6(b) and (c). FOS then removes some points so that the resultant V and U remain concave and convex, respectively, as shown in Fig. 6(c) and (d). By triangle inequality, we can prove that the shortest path is in the funnel. If the resultant U and V will cross each other, as shown in Fig. 7(a) and (c), FOS deterministically generates the transmission schedule. When there is no more frame size information available, as

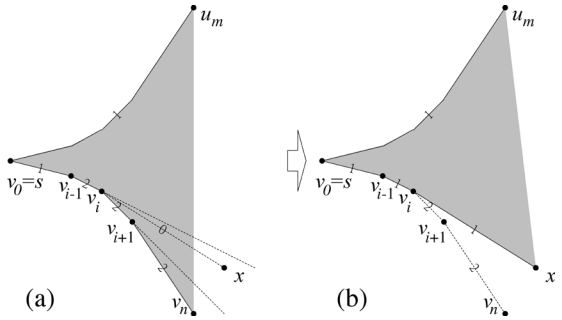


Fig. 8. Processing x. (a) FOS first scans the lower chain. If there is a point v_i such that $V' = \{v_0, \ldots, v_i, x\}$ is concave. (b) FOS modifies the funnel. In this case, FOS does not generate the transmission schedule. The figure is skewed for better visualization.

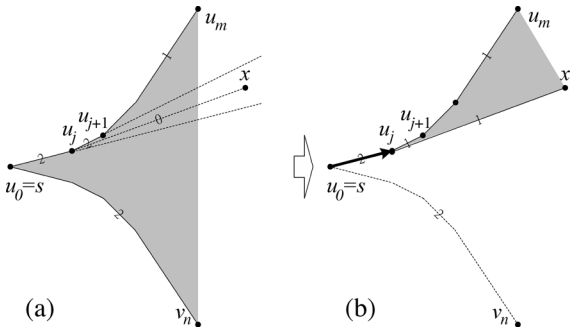


Fig. 9. Processing x. (cont.) (a) If FOS fails to find v_i, it scans the upper chain and finds a point u_j so that the edge $\overline{u_j x}$ will not cross the upper chain. (b) FOS modifies the funnel. If $j \neq 0$, FOS generates the transmission schedule according to $\{u_0, \ldots, u_j\}$. The figure is skewed for better visualization.

shown in Fig. 7(e), FOS heuristically chooses a point x'' in the funnel and generates the transmission schedule. The length of each transmission schedule is dynamic and after it is generated, FOS reconstructs the funnel, as shown in Fig. 7(b), (d), and (f). When the transmission schedule has finished and at the same time new frames have been generated, FOS computes the next transmission schedule. The detailed description of the funnel maintenance follows.

When the ath frame $(t+1 \leq a \leq t+D)$ is considered, FOS first tries to append the point $x = (a, F(a))$ onto V. As shown in Fig. 8(a), along V, FOS tries to find a point v_i $(1 \leq i \leq n)$ from v_n to v_1 such that $\mathrm{Rate}(v_{i-1}, v_i) > \mathrm{Rate}(v_i, x)$. If such a point is found, FOS removes $v_n, \ldots, v_{i+1}$ from V and adds x to the tail of V. As shown in Fig. 8(b), the resultant chain $V' = \{v_0, \ldots, v_i, x\}$ is concave. V' and U maintain the funnel. In this case, FOS does not generate the transmission schedule. If there is no such point, as shown in Fig. 9(a), along U, FOS tries to find a point u_j $(0 \leq j \leq m-1)$ from u_0 to u_{m-1} so that edge $\overline{u_j x}$ will not cross U, i.e., $\mathrm{Rate}(u_j, x) < \mathrm{Rate}(u_j, u_{j+1})$; otherwise, FOS sets $u_j = u_m$. As shown in Fig. 9(b), FOS replaces V with $V' = \{u_j, x\}$. If $j = 0$, V' and U maintain the funnel and FOS does not generate the transmission schedule. If $j \neq 0$, FOS generates the transmission schedule according to $\{u_0, \ldots, u_j\}$ and then removes $u_0, \ldots, u_{j-1}$ from U. The resultant chain $U' = \{u_j, \ldots, u_m\}$ remains convex. V' and U' maintain the funnel again.

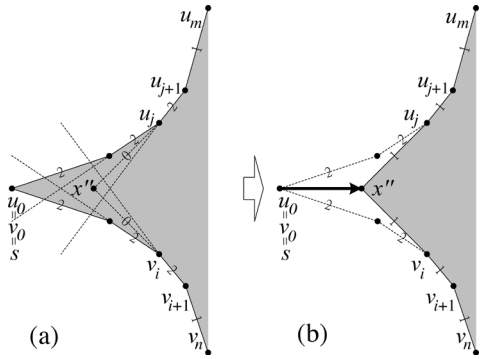


Fig. 10. Processing x''. (a) After FOS heuristically chooses a point x'' in the funnel, it finds v_i and u_j on the lower chain and the upper chain, respectively, so that $\{x'', v_i, v_{i+1}, \ldots, v_n\}$ is concave and $\{x'', u_j, u_{j+1}, \ldots, u_m\}$ is convex. (b) FOS reconstructs the funnel and computes the transmission schedule according to $\{s, x''\}$. The figure is skewed for better visualization.

After appending x onto V, FOS then appends the point $x' = (a, H(a))$ onto U. The processing of x' is similar to the processing of x. FOS tries to find a point u_j from u_m to u_1 so that $U' = \{u_0, \ldots, u_j, x'\}$ is convex. If such a point is found, U' and V maintain the funnel and FOS does not generate the transmission schedule. If there is no such point, FOS then finds a point v_i from v_0 to v_n so that $U' = \{v_i, x'\}$ and $V' = \{v_i, \ldots, v_n\}$ maintain the funnel. If $i \neq 0$, FOS generates the transmission schedule according to $\{v_0, \ldots, v_i\}$ and then removes $v_0, \ldots, v_{i-1}$ from V.

Saturation may occur if FOS does not decide the transmission schedule after the $(t+D)$th frame is considered. Since there is no more frame size information available, FOS heuristically selects a point x'' in the funnel and generates the transmission schedule according to $\{s, x''\}$, as shown in Fig. 7(e). Note that x'' must satisfy

$$r_{\min} \leq \mathrm{Rate}(s, x'') \leq r_{\max} \tag{8}$$

where

$$r_{\min} = \mathrm{Rate}(s, v_1) \tag{9}$$

and

$$r_{\max} = \mathrm{MIN}\left(\mathrm{Rate}(s, u_1), \frac{F(t+D) - G(t)}{L}\right) \tag{10}$$

i.e., $r_{\min}$ and $r_{\max}$ are the minimal and maximal feasible transmission rates from the point s. Many heuristics can be used. FOS sets $L = 1$ to incorporate new frame size information as early as possible, and uses the minimal feasible transmission rate in the funnel. It selects the point $x'' = (t+1, G(t) + r_{\min})$. FOS then reconstructs the funnel. As shown in Fig. 10, 1) along V, FOS finds a point $v_i (0 \leq i \leq n-1)$ from v_0 to v_{n-1} such that $Rate(x'', v_{i+1}) > Rate(v_i, v_{i+1})$ and thus $V' = \{x'', v_i, v_{i+1}, \ldots, v_n\}$ is concave. 2) Along U, FOS finds a point u_j $(0 \leq j \leq m-1)$ from u_0 to

u_{m-1} such that $Rate(x'', u_{j+1}) < Rate(u_j, u_{j+1})$ and thus $U' = \{x'', u_j, u_{j+1}, \ldots, u_m\}$ is convex. U' and V' maintain the funnel.

While the video server executes S^t, new frames are simultaneously generated. Assume S^t is an L frametime transmission schedule. When S^t has finished and therefore L new frames have already been generated, FOS computes the next transmission schedule S^{t+L}.

B. Complexity Analysis

We assume each edge that is added onto the funnel associates with a counter. The counter is initialized as zero and increases by one whenever the edge is scanned. The time complexity analysis of FOS is based on the following claims.

Claim A: The counter of an edge that is added onto the funnel and removed later is read as two.

Claim B: The counter of an edge that remains on the funnel is read as one.

Proof:

1) Consider the process of appending x onto V. FOS first starts scanning the edges V on from the tail. If there is a point v_i $(1 \leq i \leq n)$ such that $Rate(v_{i-1}, v_i) > Rate(v_i, x)$, the scan stops. The edges on $\{v_{i-1}, v_i, \ldots, v_n\}$ have been scanned and their associated counters increased from one to two. Then, $v_{i+1}, \ldots, v_n$ are removed and x is added to the tail of V. Note that the counter of the edge $\overline{v, x}$ is initialized as zero. We can amortize the two counters of the edges $\overline{v_{i-1}v_i}$ and $\overline{v_i x}$ so that each counter is reset to one, as shown in Fig. 8. Thus, claims A and B hold. If there is no such point, FOS starts scanning the edges on U from the head. Applying similar amortized analyses, we can prove that claims A and B also hold after the scan stops, as shown in Fig. 9. Therefore, after FOS completes the process of appending x onto V, claims A and B hold.
2) Similarly, we can prove that after FOS completes the process of appending x' onto U, claims A and B hold.
3) When there is no more frame size information available, FOS computes x'' in constant time and then reconstructs the funnel. As shown in Fig. 10, FOS scans the edges on V from the head. When FOS finds a point v_i $(0 \leq i \leq n-1)$ such that $Rate(x'', v_{i+1}) > Rate(v_i, v_{i+1})$, the scan stops. The edges on $\{v_0, \ldots, v_i, v_{i+1}\}$ have been scanned and their associated counters increased from one to two. Then, $v_0, \ldots, v_{i-1}$ are removed x'' and is added to the head of V. Note that the counter of the edge $\overline{x'' v_i}$ is initialized as zero. We can amortize the two counters of the edges $\overline{x'' v_i}$ and $\overline{v_i v_{i-1}}$ so that each counter is reset to one. Therefore, claims A and B hold. FOS then scans the edges on U from the head. Again by applying similar analyses, we can prove that after FOS completes the reconstruction, claims A and B hold. ■

FOS considers $F(a)$ and $H(a)$ once for $0 \leq a \leq N-1$. Each time, FOS adds one edge onto the funnel and may remove some edges. Since the number of transmission schedules generated will be N, at the most, FOS reconstructs the funnel at most N times. Each time, FOS may add two edges onto the funnel and remove some edges. In total, FOS adds a maximum of $4 * N$ edges onto the funnel. Thus, there is a maximum of $4 * N$ associated counters and the summation of all readings is smaller than $8 * N$. Therefore, the time complexity of FOS is $O(N)$.

C. Heuristics

Consider the three saturations in which FOS deterministically, or heuristically, computes the transmission schedule. It is easy to see that FOS computes the same transmission schedule as $SLWIN(1)$. As stated in the end of Section II, it is also true that FOS may lower the transmission rate unnecessarily and then raise the rate in the next transmission schedule. Note that when there is no more frame size information available, FOS heuristically chooses to transmit media data over a frametime at the rate $r_{\min}$. If a traffic-smoothing algorithm has aggressively transmitted more data to the client, there are less data buffered in the server. Therefore, it is likely the algorithm can smooth the upcoming large frame.

There exist different heuristics. For example, smoothing algorithms can predict the transmission rate in the near future [17], [18]. We have derived a variant algorithm called $FOS1$ that considers the current transmission rate as a simple form of rate prediction. $FOS1$ transmits media data over a frametime at the rate

$$r' = \mathrm{MAX}\left(\mathrm{MIN}(r_{\mathrm{cur}}, r_{\max}), r_{\min}\right) \tag{11}$$

where $r_{\mathrm{cur}} = r_{t-1}$ at time $t + D$.

We have also derived another variant algorithm called $FOS2$ that considers the current peak rate. $FOS2$ will try to keep the client buffer full without raising the current peak rate. It transmits media data over a frametime at the rate

$$r'' = \mathrm{MAX}\left(\mathrm{MIN}(r_{peak}, r_{\max}), r_{\min}\right) \tag{12}$$

where $r_{\mathrm{peak}} = \mathrm{MAX}(r_{-D}, \ldots, r_{t-1})$ at time $t + D$.

By definition, $r_{\mathrm{cur}} \leq r_{\mathrm{peak}}$ and thus $r_{\min} \leq r' \leq r'' \leq r_{\max}$. $FOS2$ works ahead more aggressively than $FOS1$ and FOS. The aggressive workahead has a significant impact on reducing the peak rate when the initial delay is small. In this case, there is not much time for traffic-smoothing algorithms to work ahead on the upcoming large frames. If $FOS2$ has aggressively transmitted more data to the client, there is a high probability it will smooth the large frames.

IV. Experiment Results

In this section, we examine the performance of the proposed online traffic-smoothing algorithms. The study focuses on network and client resources by measuring the peak rate and the client buffer occupancy. The peak rate of a smoothed video stream determines its worst-case bandwidth requirement across the path from the video server to the playback client. Hence, most traffic-smoothing algorithms attempt to minimize the peak rate to increase the likelihood that the video server, network and client player have sufficient resources to handle the stream.

TABLE I
SOME STATISTICS OF THE FOUR MPEG VIDEO CLIPS

Video	Number of frames	FPS	GOP	Maximum frame size (bytes)	Average frame size (bytes)	STD (bytes)
Star Wars	40,000	24	12	124,816	9,313.2	12,902.7
MTV	40,000	24	12	229,200	24,604.2	23,062.6
Talk	40,000	24	12	106,768	14,536.8	16,519.5
Soccer	40,000	24	12	190,296	25,109.6	21,260.6

FPS is the number of frames per second. GOP means Group of Picture. STD is the standard deviation of frame sizes.

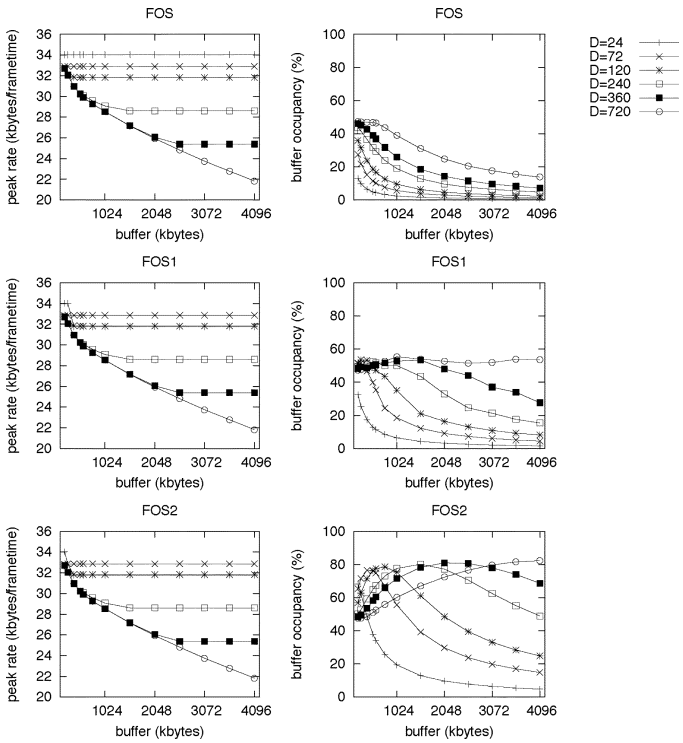


Fig. 11. Peak bandwidth and client buffer utilization of transmission schedules for Star Wars as a function of the client buffer size for variant initial delays.

Practically, media data may get lost in the network. However, if the client player can detect the data loss early enough, it can request a retransmission. A transmission schedule with a high percentage of client buffer occupancy usually implies that there is a high probability the client player will detect the data loss early.

We simulate the transmission of four 40 000-frame MPEG video clips [21]. Table I shows some statistics of these streams. First, we explore how the two metrics, the initial delay (D frametimes) and the client buffer size (B kB), change as a function. We then compare the performance of the proposed algorithms to $SLWIN(1)$ when the initial delay is small. $SLWIN(1)$ looks ahead D frames to compute the transmission schedule, i.e. $W = D$. We can see that FOS indeed computes the same transmission schedule as $SLWIN(1)$. However, the time complexity of FOS is only $1/W$ of $SLWIN(1)$. In [10] and [11], Sen *et al.* have shown that $SLWIN(1)$ consistently outperforms $SLWIN(k)$. To demonstrate the lower bound of the peak rate, the graph also plots the optimal transmission schedule obtained by the offline algorithm presented in [1].

A. Client Buffer Size and Initial Delay

Using Star Wars as the test video, Fig. 11 illustrates the performance of the three proposed algorithms FOS, $FOS1$ and $FOS2$. Generally speaking, the client buffer size limits the maximum data that can be workahead transmitted to the client. If the client buffer is small, smoothing algorithms cannot transmit media data in advance of the burst. As the client buffer is enlarged and more data can be transmitted in advance, the transmission rate can be reduced. High client buffer occupancy shows that most media data arrives at the client much earlier than its playback time. However, enlarging the client buffer beyond a certain value does not help smoothing algorithms to reduce the peak rate. When the client buffer is comparatively huge with the initial delay, the client buffer occupancy also starts falling.

When the initial delay is small, there are few frames in the server buffer. Consequently, the benefit from a large client buffer is not significant. As shown in Fig. 11, when the initial delay is smaller than 72 frametimes, using a client buffer larger than 512 kB does not help the three algorithms to further reduce the peak rate. Since there is little data to be buffered, the client buffer occupancy falls rapidly as the client buffer is enlarged. Heuristics play an important role when the initial delay is small. We will discuss this in more detail in the next subsection.

As the initial delay increases, the benefit from a large client buffer becomes apparent. A large initial delay gives the video server more time to transmit any frame, and thus the peak rate is reduced. In such cases, there is a high probability that FOS, $FOS1$, and $FOS2$ will deterministically compute the transmission schedule at any time. Heuristics, therefore, have little impact on reducing the peak rate. For example, when the initial delay is 360 or 720 frametimes, the peak rate computed by FOS, $FOS1$ and $FOS2$ is almost the same for variant client buffer sizes. Even so, aggressive workahead utilizes the client buffer more efficiently. $FOS2$ utilizes 60% to 80% of the client buffer. However, $FOS1$ only achieves 40% to 60% while FOS achieves 40% only when client buffer is smaller than 1024 kB.

When the initial delay is comparatively large with the client buffer size, the three algorithms always deterministically compute the transmission schedule. Heuristics are, therefore, never used. The three algorithms compute the same transmission schedule, for example, when the initial delay is 3600 or 7200 frametimes and the client buffer is smaller than 2048 kB.

B. Heuristics and Initial Delay

In this subsection, we further compare the performance of online traffic-smoothing algorithms when the initial delay is small. In such cases, online traffic-smoothing algorithms lack the information to deterministically compute a transmission schedule most of the time. Heuristics play an important role. Since large client buffers do not have significant impacts, we choose two client buffer sizes that are slightly larger than the double and triple of the maximal frame size. In real applications, the system

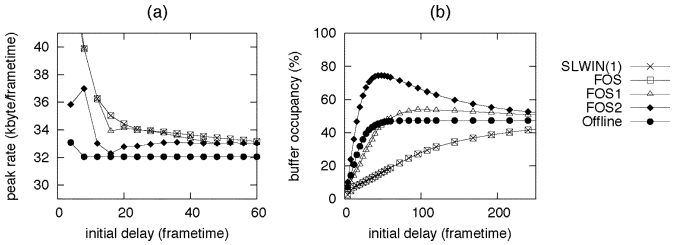


Fig. 12. Peak bandwidth and client buffer utilization of transmission schedules for Star Wars when initial delays are small. ($B = 256$ kB).

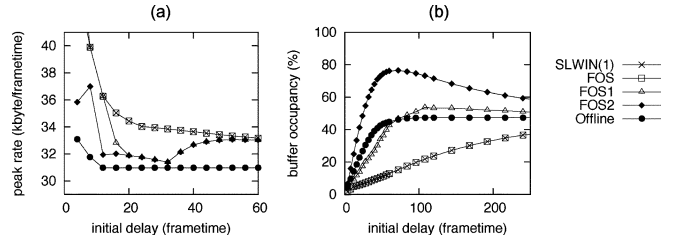


Fig. 13. Peak bandwidth and client buffer utilization of transmission schedules for Star Wars when initial delays are small. ($B = 384$ kB).

may determine this value according to experiential rules, such as the desired quality of the video. Note that $SLWIN(1)$ computes the same transmission schedule as FOS. Their performance curves are overlapped in Fig. 12 and 13.

As shown in Fig. 12 and Fig. 13, online traffic-smoothing algorithms hardly reduce the peak rate for transmitting Star Wars by enlarging the client buffer from 256 to 384 kB. $FOS2$ constantly performs better than $FOS1$ and FOS in reducing the peak rate and utilizing the client buffer. $FOS2$ dramatically reduces the peak rate when the initial delay is within 12 to 48 frametimes. It reduces the peak rate to 32 kB per frametime when the initial delay is 16 frametimes. Analyzing the frame sizes of Star Wars, there are two bursts close to one another. The second is slightly longer and burstier than the first. If the initial delay is smaller than 48 frametimes, $FOS2$ cannot smooth the first burst. It therefore raises the peak rate. Since $FOS2$ aggressively works ahead and there are less data buffered in the server, it can smooth the second burst. If the initial delay is larger than 48 frametimes, $FOS2$ is able to smooth the first burst with a smaller rate. This time, however, $FOS2$ cannot smooth the second burst and therefore the peak rate increases.

Experiments on the other video clips show similar trends. Table II gives a brief summary of the improvement of $FOS2$ on $SLWIN(1)$ in rate reduction when the initial delay is small. The results show that the aggressive workahead heuristic significantly improves the performance of smoothing algorithms, especially for real-time interactive applications.

V. Conclusion

In this paper, we first present an efficient traffic-smoothing algorithm FOS for live video transmission. Given a client buffer and a playback delay, FOS maintains the candidates for transmission schedules in a funnel. Whenever the two chains of the funnel will cross each other, FOS deterministically generates

TABLE II
Improvement of $FOS2$ on $SLWIN(1)$ in Rate Reduction

Video	Buffer size (Kbytes)	Initial delay (frametime) 8	16	24
Star	256	7.3%	7.8%	3.5%
	384	7.3%	8.6%	6.7%
MTV	512	7.0%	5.5%	2.6%
	768	7.0%	5.2%	2.3%
Talk	384	4.9%	2.1%	2.9%
	576	4.9%	2.1%	2.9%
Soccer	384	4.6%	4.4%	1.1%
	576	3.6%	5.7%	5.5%

the transmission schedule. When there is no more frame size information available, FOS heuristically generates the transmission schedule at the minimal rate for the next frametime. The total time complexity of FOS is $O(N)$. In fact, FOS generates the same transmission schedule as $SLWIN(1)$.

We observe that FOS may lower the transmission rate unnecessarily and then raise the rate in the next transmission schedule. We have used different heuristics to improve FOS. $FOS1$ considers the current transmission rate and $FOS2$ aggressively considers the current peak rate. When the initial delay is small, heuristics play an important role. Experiment results show that $FOS2$ further reduces the peak bandwidth requirement and better utilizes the client buffer for real-time interactive applications in which the initial delay is small.

In the best-effort service model, media data transmitted across the Internet are normally subject to delay jitter, out-of-order delivery, and packet loss. The working of smoothing algorithms will be seriously impaired if there are errors in the transmission medium. Many error recovery techniques for multimedia streaming have been proposed, such as automatic repeat request (ARQ), forward error correction (FEC), etc. [20]. Since $FOS2$ usually generates the transmission schedule with high client buffer occupancy, the media data should arrive at the client much earlier before its playback time. If there is an error, the client can detect the error early and request a retransmission. In such cases, the server may retransmit the data and recompute the funnel. We will further study this problem in the future.

References

[1] J. D. Salehi, Z.-L. Zhang, J. Kurose, and D. Towsley, "Supporting stored video: Reducing rate variability and end-to-end resource requirement through optimal smoothing," *IEEE/ACM Trans. Network.*, vol. 6, no. 4, pp. 397–410, Aug. 1998.

[2] W. Feng and S. Sechrest, "Critical bandwidth allocation for delivery of compressed video," *Comput. Commun.*, pp. 709–717, Oct. 1995.

[3] W. Feng, F. Jahaian, and S. Sechrest, "Optimal buffering for the delivery of compressed video," in *IS&T/SPIE Multimedia Computing and Networking (MMCN)*, San Jose, CA, 1995, pp. 234–242.

[4] R. I. Chang, M. Chen, M. T. Ko, and J. M. Ho, "Optimization of stored VBR video transmission on CBR channel," in *SPIE Voice, Video, and Data Communications (VVDC) Symp.*, 1997, pp. 382–392.

[5] R. I. Chang, M. Chen, M. T. Ko, and J. M. Ho, "Designing the on-off CBR transmission schedule for jitter-free VBR media playback in real-time networks," in *Proc. IEEE Real-Time Computinmg Systems and Applications (RTCSA) Conf.*, 1997, pp. 1–9.

[6] J. M. McManus and K. W. Ross, "Video on demand over ATM: constant-rate transmission and transport," in *Proc. IEEE INFOCOM*, San Francisco, CA, Mar. 1996.

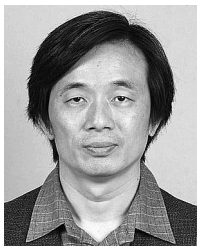
[7] J. M. McManus and K. W. Ross, "Dynamic programming methodology for managing prerecorded VBR sources in packet-switched networks," in *SPIE Voice, Video, and Data Communications (VVDC) Symp.*, Univ. Pennsylvania, 1997.
[8] M. Grossglauser, S. Keshav, and D. Tse, "RCBA: a simple and efficient service for multiple time-scale traffic," in *Proc. ACM SIGCOMM*, Aug. 1995, pp. 219–230.
[9] W. Feng and J. Rexford, "A comparison of bandwidth smoothing techniques for the transmission of prerecorded compressed video," in *Proc. IEEE INFOCOM*, Apr. 1999, pp. 58–66.
[10] J. Rexford, S. Sen, J. Dey, W. Feng, J. Kurose, J. Stankovic, and D. Towsley, "Online smoothing of live, variable-bit-rate video," in *Proc. International Workshop on Network and Operating Systems Support for Digital Audio and Video*, May 1997, pp. 249–257.
[11] 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.
[12] R. I. Chang, M. C. Chen, J. M. Ho, and M. T. Ko, "An effective and efficient traffic smoothing scheme for delivery of online VBR media streams," in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 447–454.
[13] G. Cao, W. Feng, and W. Singhal, "Online VBR video traffic smoothing," in *Proc. 8th IEEE Int. Conf. Computer and Communication and Networks*, Oct. 1999, pp. 502–507.
[14] D. T. Lee and F. P. Preparata, "Euclidean shortest path in the presence of rectilinear barriers," *Networks*, vol. 14, pp. 393–410, 1984.
[15] M. Garrett and W. Willinger, "Analysis, modeling and generation of self-similar VBR video traffic," in *Proc. ACM SIGCOMM*, London, U.K., Sep. 1994.
[16] M. Krunz and S. K. Tripathi, "On the characteristics of VBR MPEG streams," in *Proc. ACM SIGMETRICS*, Jun. 1997, pp. 192–202.
[17] A. Ads, "Supporting real-time VBR video using dynamic reservation based on linear prediction," in *Proc. IEEE INFOCOM*, Mar. 1996, pp. 1476–1483.
[18] S. Crosby, M. Huggard, I. Leslie, J. Lewis, B. McGurk, and R. Russell, "Predicting bandwidth requirement of ATM and ethernet traffic," in *Proc. IEE UK Teletraffic Symp.*, Manchester, U.K., Mar. 1996.
[19] E. Amir, S. McCanne, and H. Zhang, "An application level video gateway," in *Proc. ACM Multimedia*, San Francisco, CA, Nov. 1995.
[20] G. Carle and E. W. Biersack, "Survey of error recovery techniques for IP-based audio-visual multicast applications," *IEEE Network*, vol. 11, no. 6, pp. 24–36, Nov./Dec. 1997.
[21] [Online]. Available: http://www3.informatik.uni-wuerzburg.de/MPEG/traces/

Jeng-Wei Lin received the B.S. degree in computer information science from National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1994, and the M.S. and Ph.D. degrees in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 1996 and 2005, respectively.

He joined the Department of Information Management, Tunghai University, Taichung, Taiwan, in 2005. From 1996 to 2005, he was a Research Assistant with the Institute of Information Science, Academia Sinica, Taipei, and was then promoted to Postdoctoral Researcher. His research interests include multimedia systems, traffic management, and other areas in networking.

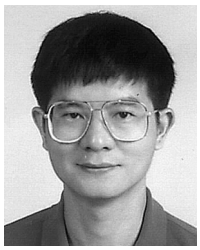
Ray-I Chang (M'96) received the Ph.D. degree in electrical engineering and computer science from National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1996.

He then joined the Computer Systems and Communications (CSCL) Laboratory, Institute of Information Science, Academia Sinica. In 2003, he joined the Department of Engineering Science, National Taiwan University, Taipei.



Jan-Ming Ho (M'90) received the Ph.D. degree in electrical engineering and computer science from Northwestern University, Evanston, IL, in 1989 and the B.S. degree in electrical engineering from National Cheng-Kung University, Tainan, Taiwan, R.O.C., in 1978, and the M.S. degree from the Institute of Electronics, National Chiao-Tung University, Hsinchu, Taiwan, in 1980.

He joined the Institute of Information Science, Academia Sinica, Taipei, Taiwan, as an Associate Research Fellow in 1989, and was promoted to Research Fellow in 1994. He visited IBM T. J. Watson Research Center, Yorktown Heights, NY, in the summers of 1987 and 1988, the Leonardo Fibonacci Institute for the Foundations of Computer Science, Trento, Italy, in the summer of 1992, and the Dagstuhl-Seminar on "Combinatorial Methods for Integrated Circuit Design", IBFI-Geschäftsstelle, Schloß Dagstuhl, Fachbereich Informatik, Universität des Saarlandes, Germany, in October 1993. He is currently Director General, Division of Planning and Evaluation, National Science Council, Taiwan. His research interests include web mining, video streaming, and combinatorial optimization.



Feipei Lai (SM'92) received the B.S.E.E. degree from National Taiwan University (NTU), Taipei, Taiwan, R.O.C., in 1980, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign in 1984 and 1987, respectively.

He is a Professor in the Department of Electrical Engineering and in the Department of Computer Science and Information Engineering, NTU. He is also the Director of the Computer and Information Network Center at NTU, and the Vice Superintendent of NTU Hospital. His current research interests are low-power circuit design, high-performance microprocessor chip design, computer architecture, optimizing compilers, VLSI design, MPEG4 design, and cryptography.

Dr. Lai is included in *Who's Who in Science and Engineering* and *Who's Who in the World*.