Aggressive Traffic Smoothing for Delivery of Online Multimedia*

Jeng-Wei Lin[1,3], Ray-I Chang[2], Jan-Ming Ho[1], and Feipei Lai[3,4]

[1] Institute of Information Science, Academia Sinica, Taipei, Taiwan
{jwlin, hoho}@iis.sinica.edu.tw
[2] Dept. of Engineering Science and Ocean Engineering,
[3] Dept. of Computer Science and Information Engineering, and
[4] Dept. of Electrical Engineering, National Taiwan University, Taipei, Taiwan
{rayichang, flai}@ntu.edu.tw

Abstract. Traffic smoothing is an efficient means to reduce the bandwidth requirement for transmitting a VBR video. For live video applications, Sen et al. present an online algorithm referred to as $SLWIN(k)$ to compute the transmission schedule on the fly. $SLWIN(k)$ looks ahead W 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 playback. 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 denoted as ATS (Aggressive Traffic Smoothing). ATS aggressively works ahead to transmit more data as early as possible for reducing the peak rate of the bandwidth requirement. We compare the performance of ATS with $SLWIN(k)$ based on several benchmark video clips. Experiment results show that ATS further reduces the bandwidth requirement, especially for interactive applications in which the initial delays are small.

1 Introduction

A growing number of applications such as digital libraries and newscasts require real-time multimedia to be accessed on networks. For continuous playback, the client player must render a new frame after one *frametime* has passed. To prevent the client from starvation, the video server has to transmit the video data into the client buffer before it is going to be rendered. However, video streams are compressed and often exhibit significant burstiness of *frame sizes* on many time scales due to the encoding frame structure and their natural variations within and between scenes [12][13]. This burstiness complicates the design of a multimedia system for high resource utilization, such as network bandwidth and the client buffer.

For a prerecorded video, the video server knows 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

* This paper was partially supported by NSC, Taiwan, under grants NSC91-2416-H-008-008-, NSC92-2416-H-002-051-, and NSC93-2213-E-002-086-.

K. Aizawa, Y. Nakamura, and S. Satoh (Eds.): PCM 2004, LNCS 3331, pp. 114–121, 2004.

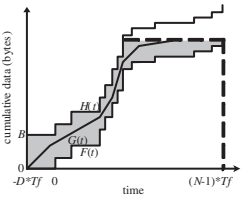

Fig. 1. A feasible transmission schedule.

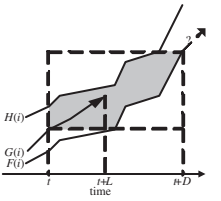

Fig. 2. Looking ahead in a live video.

the burst. By working ahead, traffic smoothing is proved efficient at reducing the bandwidth requirement for VBR video transmission [1]-[9]. In live video applications, however, the server has limited knowledge of frame sizes at any one time. Sen et al. introduce a sliding-window traffic smoothing algorithm referred to as $SLWIN(k)$ that computes the transmission schedule on the fly [11]. Given an initial delay, $SLWIN(k)$ looks ahead a window of W frames and uses an offline traffic smoothing algorithm to compute the transmission schedule for the next k frametimes, where $k \leq W$. Note that W is upper bounded by the initial delay. For an N frame live video, the time complexity of $SLWIN(k)$ is $O(W * N/k)$. In this paper, we present an $O(N)$ traffic smoothing algorithm denoted as ATS (Aggressive Traffic Smoothing) that online computes the transmission schedule for live video applications. ATS generates the transmission schedule that works ahead as aggressively as possible without raising the peak rate of the transmission schedule. Experiment results show that ATS further reduces the bandwidth requirement and utilizes the client buffer more efficiently.

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 2. Section 3 presents our ATS algorithm and its time complexity proof. Section 4 shows the experiment results. Finally, in section 5, we give conclusions.

2 Online Traffic Smoothing for Live Video

For an N frame video $V=\{f_0, f_1, f_2, \ldots, f_{N-1}; T_f\}$, which uses f_i bits to encode the i-th frame and T_f is the frametime, a playback schedule can be represented as a cumulative playback function (CPF) $F(t) = F_i$ for $i * T_f \leq t < (i+1) * T_f$, where $F_i = 0$ for $i < 0$ and $F_i = F_{i-1} + f_i$ for $0 \leq i \leq N-1$. 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, the video server can transmit media data at 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 (CTF) is defined as $G(t) = 0$ for $t \leq -D * T_f$ and $G(t) = G(i * T_f) + r_i * (t - i * T_f)$ for $i * T_f < t \leq (i+1) * T_f$. To guarantee continuous playback at the client, S should satisfy $F(t) \leq G(t)$ for $t \leq (N-1) * T_f$. We assume the client provides a B-bit buffer. The cumulative buffer function (CBF) is defined as $H(t) = F_{i-1} + B$ for $i * T_f < t \leq (i+1) * T_f$. To avoid

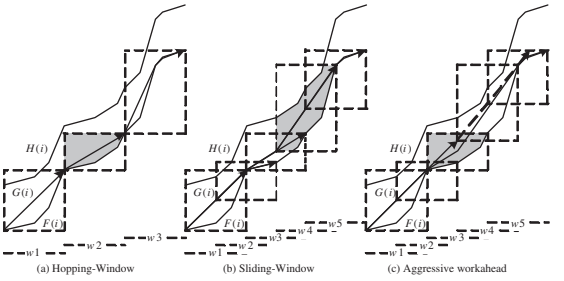

Fig. 3. Smoothing across window boundaries.

buffer overrun, S should also satisfy $G(t) \leq H(t)$ for $t \leq (N-1) * T_f$, as shown in Fig. 1. 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.

$$
\begin{array}{lll}
CPF: & F(i) = 0, & i < 0 \\
 & F(i) = F(i-1) + f_i, & 0 \leq i \leq N-1 \\
CBF: & H(i) = F(i-1) + B, & i \leq N-1 \\
CTF: & G(i) = 0, & i \leq -D \\
 & G(i) = G(i-1) + r_{i-1}, & -D < i \leq N-1
\end{array}
$$

For a prerecorded video stream, a traffic smoothing algorithm knows each frame size and can offline compute the transmission schedule such that $F(i) \leq G(i) \leq Min(H(i), F(N-1))$ for $-D \leq i \leq N-1$. For live video, however, a traffic smoothing algorithm has limited knowledge of frame sizes because future frames have not been generated. At any one time $t + D$, the algorithm knows f_{t+1}, $f_{t+2}, \ldots, f_{t+D}$ and has no idea about f_{t+D+1}, $f_{t+D+2}, \ldots, f_{N-1}$. Any L ($1 \leq L \leq D$) frametime transmission schedule $S^t = \{r_t, r_{t+1}, \ldots, r_{t+L-1}\}$ is feasible if S^t satisfies $F(i) \leq G(i) \leq Min(H(i), F(t+D))$ for $t < i \leq t + L$, as shown in Fig. 2. 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}.

SLWIN(k) fixes L to a pre-assigned constant k ($k \leq W$). In this paper, we assume $W = D$ so that *SLWIN*(k) looks ahead as more frames as possible. When $k = W$, *SLWIN*(W), also referred to as the hopping-window approach, is fast. However, it has drawbacks of smoothing traffic across window boundaries, as shown in Fig. 3(a). Intuitively, the bandwidth allocated for transmitting each frame should be decided with as much information as possible. As shown in Fig. 3(b), when using a smaller k, *SLWIN*(k) computes the transmission schedule of a smaller bandwidth requirement. Sen et al. showed that *SLWIN*(1) outperforms other *SLWIN*(k) algorithms [11]. However, *SWLIN*(1) computes the transmission schedule independently for each window. Therefore, the computing cost is large. In addition, $H(i)$ is suppressed by $F(t + D)$, as shown in Fig. 2. *SLWIN*(k) may lower the transmission rate unnecessarily since it always selects the smallest bandwidth requirement for each window. As shown in Fig. 3(b),

the transmission rate decreases in the third window and then increases in the fourth window. However, if the video server aggressively transmits more data in the third window, the bandwidth requirement for the fourth window can be reduced, as shown in Fig. 3(c). Since the server works ahead more aggressively, less data is buffered in the server. Therefore, the server can use a lower transmission rate to transmit the upcoming large frame.

3 Our Algorithm and Time Complexity Analysis

Like *MVBA* [1] (the offline smoothing algorithm used by $SLWIN(k)$), *ATS* tries to find the shortest path between $F(i)$ and $MIN(H(i), F(i+D))$. To reduce the suppression effect, *ATS* heuristically generates the transmission schedules that work ahead as aggressively as possible without raising the peak rate. Since the suppression will be looser after a new frame is generated, aggressive workahead lasts for one frametime. To minimize the computing cost, *ATS* processes each frame once. It remembers useful computational results for the next window by the help of a funnel data structure.

As shown in Fig. 4(a), from the starting point $s=(t, G(t))$ of the window, *ATS* maintains the candidates for transmission schedules within a convex upper chain $U=\{u_0 = s, u_1, \ldots, u_m\}$ and a concave lower chain $V=\{v_0 = s, v_1, \ldots, v_n\}$. $u_1, \ldots, u_m$ are on H and $v_1, \ldots, v_n$ are on F. Note that U is convex if and only if $0 \leq m < 2$ or $Rate(u_j, u_{j+1}) < Rate(u_{j+1}, u_{j+2})$ for $0 \leq j < m-2$, where $Rate(x, y)$ is the slope of the line from x to y. U and V form a funnel. For $t+1 \leq a \leq t+D$, *ATS* iteratively considers unprocessed $F(a)$ and $H(a)$ once and modifies V and U, respectively, as shown in Fig. 4(b) and (c). By triangle inequality, we can prove that the shortest path is in the funnel.

ATS first considers to append the point $x=(a, F(a))$ onto V. *ATS* may remove some points from V so that the resultant chain V' is still concave. If V' will not cross U, as shown in Fig. 4(b), *ATS* continues processing $H(a)$. If V' will cross U, as shown in Fig. 5(a), *ATS* deterministically generates the transmission schedule according to the line segments on U that are under the dashed line (s, x) and then maintains the funnel, as shown in Fig. 5(b). It is easy to prove these line segments are parts of the shortest path. The detail procedure follows.

As shown in Fig. 6(a), along V, *ATS* tries to find a point v_i from v_n to v_1 so that $Rate(v_{i-1}, v_i) > Rate(v_i, x)$. If such a point is found, *ATS* removes $v_{i+1}, \ldots, v_n$ from V and adds x to the tail of V. As shown in Fig. 6(b), the resultant chain $V'=\{v_0, \ldots, v_i, x\}$ is concave. V' and U maintain the funnel. In this case, *ATS* does not generate the transmission schedule. If there is no such point, along U, *ATS* tries to find a point u_j from u_0 to u_m so that the edge $u_j x$ will not cross U, as shown in Fig. 6(c). *ATS* replaces V with $V'=\{u_j, x\}$. If $j \neq 0$, *ATS* generates the transmission schedule according to the chain $\{u_0, \ldots, u_j\}$ and then removes $u_0, \ldots, u_{j-1}$ from U. The resultant chain $U'=\{u_j, \ldots, u_m\}$ is convex. V' and U' maintain the funnel again, as shown in Fig. 6(d). □

ATS then considers to append $x'=(a, H(a))$ onto U if $H(a) \leq F(t+D)$. Similarly, *ATS* may remove some points from U so that the resultant chain U' is convex. If U' will not cross V, as shown in Fig. 4(c), *ATS* continues processing

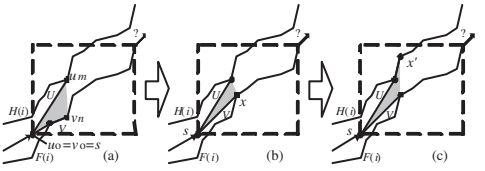

Fig. 4. *ATS* iteratively considers $F(i)$ and $H(i)$ to maintain the two chains.

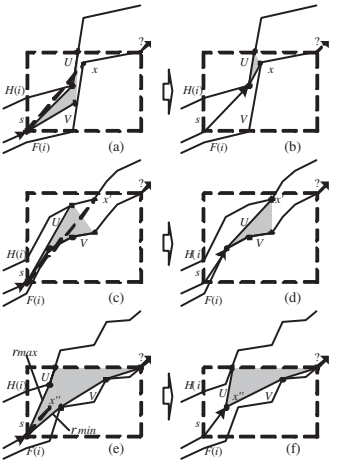

Fig. 5. Generating the schedule.

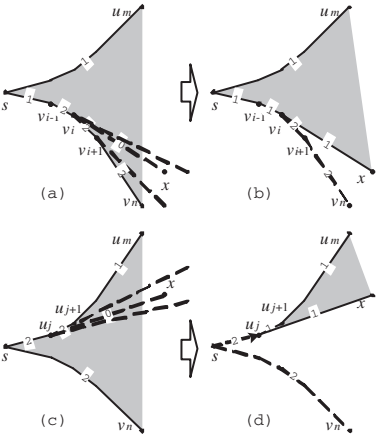

Fig. 6. Processing x.

the next frame. If U' will cross V, as shown in Fig. 5(c), *ATS* deterministically generates the transmission schedule according to the line segments on V that are above the dashed line (s, x') and then maintains the funnel, as shown in Fig. 5(d). Again, we can prove these line segments are parts of the shortest path. We omit the detail procedure since it is similar to the process of x.

Saturation may occur if *ATS* does not decide the transmission schedule after the $(t + D)$-th frame is considered, as shown in Fig. 5(e). Unlike *MVBA*, which always generates the transmission schedule according to V, *ATS* uses the aggressive workahead scheme to reduce the suppression effect. It generates the transmission schedule according to the dashed line (s, x''), where $x''=(t+1, G(t) + MAX(MIN(r_{peak}, r_{max}), r_{min}))$, r_{min} and r_{max} are the minimal and maximal feasible transmission rates from the point s and r_{peak} is the current peak rate. Note that x'' is definitely in the funnel and the length of the transmission schedule is one frametime. *ATS* then reconstructs the funnel again by removing some points and adding x'' at the head of V and U, as shown in Fig. 5(f). We omit the detail procedure again.

The primary factor of the *ATS* time complexity is the funnel maintenance. We assume each edge on the funnel associates with a counter. The counter is initialized as zero and increases by one whenever the edge is scanned.

Lemma 1. *The counter of an edge that is added onto the funnel and removed later is read as two. The counter of an edge that remains on the funnel is read as one.*

Proof. We first consider the process of appending x onto V. *ATS* starts scanning the edges of V 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 of the chain $\{v_i, \ldots, v_n\}$ have been scanned and removed and their associated counters increased by one to two. The point x is then added to the tail of V. At this point, the readings of the two associated counters on the chain $\{v_{i-1}, v_i, x\}$ are two and zero. We can amortize the two counters so that each counter is reset to one, as shown in Fig. 6(a) and (b). If there is no such point, *ATS* starts scanning the edges of U from the head, as shown in Fig. 6(c) and (d). Again, we can apply similar amortized analyses. Therefore, we prove that lemma 1 holds after *ATS* completes the process of appending x onto V. It is easy to extend this proof and show that lemma 1 holds after *ATS* ends. □

Lemma 2. *ATS adds a maximum of* $4 * N$ *edges onto the funnel.*

Proof. ATS iteratively considers $F(i)$ and $H(i)$ once for $0 \leq i \leq N-1$. Each time, *ATS* adds one edge onto the funnel and may remove some edges. Since the number of transmission schedules generated will be N, at the most, *ATS* reconstructs the funnel at most N times. Each time, *ATS* may add two edges onto the funnel and remove some edges. In total, *ATS* adds a maximum of $4 * N$ edges onto the funnel. □

Theorem 1. *The time complexity of ATS is* $O(N)$.

Proof. It can be derived from lemma 1 and 2. □

4 Experiment Results

In this section, we examine the performance of *ATS*. The study focuses on network and client resources by measuring the peak rate and buffer occupancy. The peak rate of a transmission schedule determines its worst-case bandwidth requirement across the path from the video server to the client player. Practically, data packets may get lost in the network. If the client can detect the data lost early enough, it can request a retransmission. A transmission schedule that has a high occupancy of the client buffer usually implies that there is a high probability the client will detect the data lost early.

We simulated the transmission of several MPEG video clips [14] to compare the performance of *ATS* and *SLWIN*(1). Fig. 7 and 8 show the peak rates and buffer occupancies of the schedules for transmitting Star Wars and News, respectively, with different initial delays and client buffer sizes. To demonstrate the lower bound of the peak rate, these figures also show the optimal offline schedules obtained by [1]. When the initial delay (D) is small, *ATS* is more likely to use the aggressive workahead scheme to generate the transmission schedule. As shown in Fig. 7 and 8, the peak rates of the *ATS* schedules are significantly smaller than *SLWIN*(1) when D is smaller than 30 frametimes. The aggressive workahead scheme successfully reduces the suppression effect. Since there is not much data, the buffer occupancies are around 30% to 50%. As D increases, the

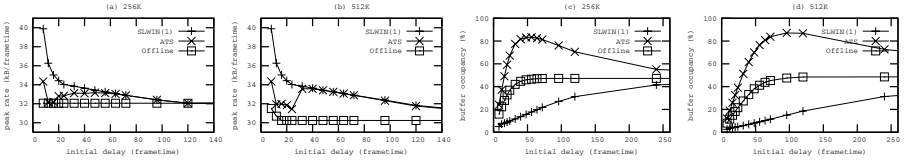

Fig. 7. Star Wars, B=256K and 512K.

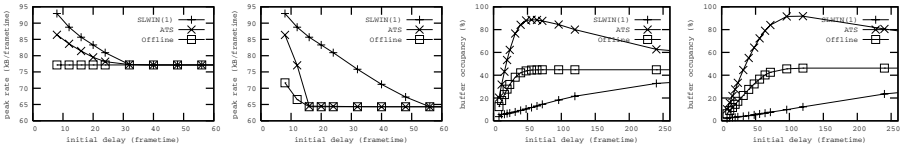

Fig. 8. News, B=512K and 1024K.

ATS schedules may have the same peak rates as $SLWIN(1)$. The buffer occupancies of *ATS* schedules keep increasing to 80% or more and then slowly decrease back to the offline optimal. When D increases beyond a certain point, *ATS* always deterministically generates the transmission schedule in every window. In such situation, *ATS* generates the same transmission schedule as $SLWIN(1)$. When *ATS* works on Star Wars, an interesting phenomenon occurs, as shown in Fig. 7(a) and (b). *ATS* dramatically reduces the peak rate to around 32 Kbytes per frametime when D is between 12 and 32 frametimes. Analyzing Star Wars, we find there are two bursts close to one another. The second is slightly burstier than the first. When D is smaller than 32 frametimes, *ATS* cannot smooth the first burst. *ATS* therefore raises the peak rate. Since *ATS* aggressively works ahead and keeps less data in the video server, it can smooth the second burst. When D is larger than 32 frametimes, *ATS* is able to smooth the first burst without raising the peak rate. However, *ATS* cannot smooth the second burst even by using the current peak rate. Therefore, the peak rate increases.

When the initial delay is small, enlarging the client buffer does not help $SLWIN(1)$ reduce the peak rate. The suppression effect eliminates the benefits from a larger buffer. As shown in Fig. 7 and 8, when the client buffer size doubles, the peak rates of the $SLWIN(1)$ schedules further decrease only when D is larger than 120 and 30 frametimes, respectively. On the other hand, the aggressive workahead scheme uses the buffer more efficiently. The peak rates of the *ATS* schedules are further reduced even D is small.

5 Conclusion

In this paper, we present an efficient traffic smoothing algorithm *ATS* for live video transmission. Unlike $SLWIN(k)$, which computes the smallest bandwidth requirement for each window independently, after *ATS* generates the transmission schedule for the current window, it remembers useful computational results for the next window by the help of a funnel data structure. The total time com-

plexity of *ATS* is $O(N)$. Note that $O(N)$ is a trivial lower bound to online smooth an N frame live video. To reduce the suppression effect caused by the lake of future frame sizes, *ATS* heuristically uses the aggressive workahead scheme to generate the transmission schedule. We have evaluated *ATS* by transmitting several benchmark video clips. Experiment results show that *ATS* further reduces the peak bandwidth requirement and better utilizes the client buffer, especially for interactive applications in which the initial delay is small.

References

1. James D. Salehi, Zhi-Li Zhang, Jim Kurose, Don Towsley, "Supporting Store Video: Reducing Rate Variability and End-to-End Resource Requirement Through Optimal Smoothing," *IEEE/ACM Trans. Networking*, Vol. 6, No.4, Aug. 1998.
2. W. Feng and S. Sechrest, "Critical Bandwidth Allocation for Delivery of Compressed Video," *Computer Communications*, pp. 709-717, Oct. 1995.
3. W. Feng, F. Jahaian and S. Sechrest, "Optimal Buffering for the Delivery of Compressed Video," *IS&T/SPIE MMCN*, pp. 234-242, 1995.
4. R. I. Chang, M. Chen, M. T. Ko and J. M. Ho, "Optimization of stored VBR video transmission on CBR channel," *SPIE, VVDC*, pp. 382-392, 1997.
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," *IEEE RTCSA*, pp. 1-9, 1997.
6. J. M. McManus and K. W. Ross, "Video On Demand over ATM: Constant-Rate Transmission and Transport," *IEEE INFOCOM*, March 1996.
7. J. M. McManus and K. W. Ross, "Dynamic Programming Methodology for Managing Prerecorded VBR Sources in Packet-Switched Networks," *SPIE VVDC*, 1997.
8. M. Grossglauser, S. Keshav and D. Tse, "RCBA: A Simple and Efficient Service for Multiple Time-Scale Traffic," in *Proc. ACM SIGCOMM*, pp 219-230, Aug. 1995.
9. Wuchi Feng, Jennifer Rexford, "A Comparison of Bandwidth Smoothing Techniques for the Transmission of Prerecorded Compressed Video," in *Proc. IEEE INFOCOM*, pp. 58-66, April 1999.
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*, pp. 249-257, May 1997.
11. Subhabrata Sen, Jennifer L. Rexford, Jayanta K. Dey, James F. Kurose, Donald F. Towsley, "Online Smoothing of Variable-Bit-Rate Streaming Video," *IEEE Trans. Multimedia*, Vol. 2, No.1, March 2000.
12. M. Garrett and W. Willinger, "Analysis, Modeling and Generation of Self-similar VBR Video Traffic," in *Proc, ACM SIGCOMM*, Sep. 1994.
13. M. Krunz and S. K. Tripathi, "On the Characteristics of VBR MPEG Streams," in *Proc. ACM SIGMETRICS*, pp. 192-202, June 1997.
14. http://www3.informatik.uni-wuerzburg.de/MPEG/traces/.