A Timestamp-Sensitive Scheduling Algorithm for MPEG-II Multiplexers in CATV Networks

Ying-Dar Lin and Chun-Mo Liu
Department of Computer and Information Science
National Chiao Tung University
Hsinchu, Taiwan
ydlin@cis.nctu.edu.tw

Abstract—**To achieve smooth display of MPEG-II programs in the residential cable TV networks, we present a timestamp-sensitive scheduling algorithm for MPEG-II multiplexers. The deadline-driven scheduler maintains, for each program stream, a counter and a timestamp to record and determine how many Transport Stream (TS) packets should be transmitted before the current scheduling cycle ends. The decoding timestamp (DTS) of TS packets is used to update the counter in order to prevent deadline violation. This algorithm is compared numerically with the timestamp-insensitive algorithm which runs constant-bit-rate (CBR) scheduling. The trace-driven simulation shows that the deadline violation of our timestamp-sensitive scheduling is much lower than CBR's and well controlled for programs with various degrees of burstiness. We also show that the algorithm can be further improved by adding a scheme to prevent buffer underflow and overflow at multiplexers and set-top-boxes, respectively.**

Keywords—**timestamp-sensitive, deadline, scheduling, MPEG-II, multiplexer, CATV**

1. Introduction

In digital video distribution systems, such as Video-On-Demand (VOD) [1] systems, the video programs are compressed into MPEG-II files which are variable-bit-rate (VBR) streams [2][3]. When the VBR streams are transmitted using the CBR service, either delay jitter or bandwidth wastage may occur. Delay jitter causes poor quality of program display. In order to prevent jitter, VBR streams can be multiplexed to share the fixed channel bandwidth, where each stream is dynamically allocated demanded bandwidth. This method, called statistical multiplexing [4][5][6], enables VBR streams to effectively share channel bandwidth because bit rates of the VBR streams are likely to compensate each other. However, the total bit rate may still exceed the channel capacity at some points in time, which may result in deadline violation and packet loss.

Some previous works solve this problem by the adaptive technique, called source rate control, to feedback control the output rate of VBR encoders [5][7][8][9]. An MPEG encoder can control, according to the feedback from the network, its output rate by setting the quantizer scale in the slice header and the optional quantizer scale in the header of each macroblock [10]. A coarser setting would result in a lower bit rate at the expense of degraded visual quality. Additionally, the encoder can also lower its output rate by discarding some of the high-frequency discrete-cosine-transform (DCT) coefficients. The schemes to reduce the output rate of an encoder discard some compressed information and are said to be lossy. This may result in visible artifacts in the decoded video.

The deadline violation and packet loss usually happens when many encoders output the intraframe compressed frames with high bit rates to the multiplexer at the same time and the total output bit rate exceeds the channel capacity. There are methods to reduce deadline violation and packet loss by transmitting the smaller interframe compressed frames in a shorter time, i.e. at a higher bit rate, and transmitting the larger intraframe compressed frames in a longer time, i.e. at a lower bit rate. This is called smoothing [3]. By smoothing the VBR MPEG compressed video, it does not need to discard compressed information. Therefore, it is lossless smoothing. Lam et al proposed an algorithm for lossless smoothing of MPEG video in [3]. The algorithm is designed to satisfy a delay bound, D, which is a pre-specified parameter. The algorithm is characterized by two other parameters, K, the number of complete pictures buffered in the queue before the multiplexer can begin sending the next picture, and H, a lookahead interval for lowering the bit rate burstiness. The objective of this lossless algorithm is to transmit each picture in the same lookahead interval at approximately the same rate, while ensuring that the buffering delay introduced by the algorithm is bounded by D for every picture.

Although the lossless algorithm can achieve good smoothing performance, it only focuses on smoothing an individual compressed video stream. It does not consider the situation where many VBR streams are transmitted in a channel with fixed bandwidth. In this paper, we consider the situation that an MPEG-II multiplexer in a cable TV network multiplexes multiple VBR MPEG-II streams. We design a timestamp-sensitive scheduling algorithm to smooth input streams, without source rate control which may degrade the video quality,

Publisher Item Identifier S 0018-9316(98)08206-7

in order to prevent deadline violation and packet loss. Our algorithm takes the advantages of statistical multiplexing and lossless smoothing. The decoding timestamp (DTS) of TS packets are used to determine how long packets can be buffered at the multiplexer. When the total bit rate exceeds the channel capacity, the scheduler in the multiplexer may transmit the TS packets whose deadlines are close by, while delaying the others. By a trace-driven simulation, it is demonstrated that our algorithm can achieve significantly lower delay, i.e. deadline violation probability, and delay jitter than the timestamp-insensitive CBR scheduling algorithm.

In [4], two methodologies, look-ahead and feedback, used for real-time rate control are presented. The look-ahead approach anticipates the bit rate that will be required by an encoder for a specific frame, but it needs preprocessing to compute the statistics to guide the allocation. The feedback approach uses previously collected statistics from the encoder to decide how many bits should be allocated to the incoming frame. The feedback approach does not need a preprocessor like the look-ahead approach, but it cannot react as fast as the look-ahead approach to changes in scene complexity. Our algorithm decides how many bits to be allocated to each input stream by checking the timestamp in real time. It does not need a preprocessor like the look-ahead approach in [4]. However, it still can precisely decide the bandwidth actually needed by the individual streams, which cannot be achieved by the feedback approach in [4].

The rest of the paper is organized as follows. Section 2 gives a background on timestamps and packet formats in MPEG-II streams. The timestamp-insensitive CBR scheduling algorithm and the timestamp-sensitive scheduling algorithm are presented in section 3. Numerical results are given in section 4. Finally, section 5 concludes the paper.

2. MPEG-II system layer specification

The ISO/IEC 13818 specification [10][11] is divided into two layers. One is the compression layer which includes video and audio parts. The other is the system layer. The compression layer focuses on how to compress the original video and audio data and specifies the syntax to represent the compressed data. The system layer addresses the combining of one or more elementary streams of video and audio, as well as other data. Into single or multiple streams which are suitable for storage or transmission. It specifies syntactical and semantic rules and provides information to enable synchronized decoding and presentation of multiplexed audio and video data. In this work, we only need the information of the system layer to develop our scheduling algorithm.

The MPEG-II system layer specifies two forms of system streams. One is Transport Stream (TS), another is Program Stream (PS). Our algorithm is designed for TS. Both kinds of streams multiplex elementary streams and are individually optimized for a different set of applications. The elementary stream is a generic form for one of the coded video, coded audio or other coded bit streams in Packetized Elementary Stream (PES) packets. Transport streams are tailored for communicating or storing one or more programs of coded data in environments in which significant errors may occur, while program streams are ailored for communicating or storing one program of coded data in environments where errors are very unlikely. Both streams provide the coding syntax which is necessary and sufficient to synchronize the decoding and presentation of the video and audio information. The syntax uses three timestamps, including Decoding Time Stamp (DTS) and Presentation Time Stamp (PTS), which concern the decoding and presentation of audio and visual data, and Program Clock Reference (PCR) or System Clock Reference (SCR), which concerns the delivery of the data stream itself.

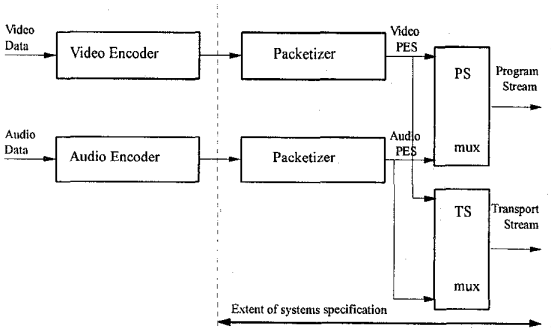

Fig. 1. Simplified overview of MPEG-II system stream [11]

Fig. 1 illustrates how the MPEG-II system stream is constructed from the visual and audio data. The visual and audio data are individually packetized to elementary streams which consist of PES packets of variable sizes. The packetizers also stamp DTS and PTS onto PES packets. The PES packets of an elementary stream are further divided into smaller Transport Stream (TS) packets of fixed 188 bytes with a specific PID (Program ID) value which appears in the TS header. Note that all TS packets of the same elementary stream have the same PID value. TS packets of different elementary streams have different PID values. The TS or PS multiplexer then multiplexes the audio and video elementary streams according to the data rates they need.

However, the multiplexer in the VOD system, as shown in Fig. 2, may be more complicated. It may contain two phases of multiplexers. The first phase multiplexer, as mentioned above, is responsible of multiplexing several elementary streams of a MPEG-II program into a single-program transport stream. The second phase multiplexer is responsible of multiplexing several MPEG-II programs and private data service into

a transport stream. The input streams of the second phase multiplexer can be either produced in real time by the first phase multiplexer preceded by a real-time MPEG-II encoder, or produced off-line beforehand and transmitted on-demand to the second phase multiplexer when subscribed. Our scheduling algorithm is to be applied to the second phase multiplexer.

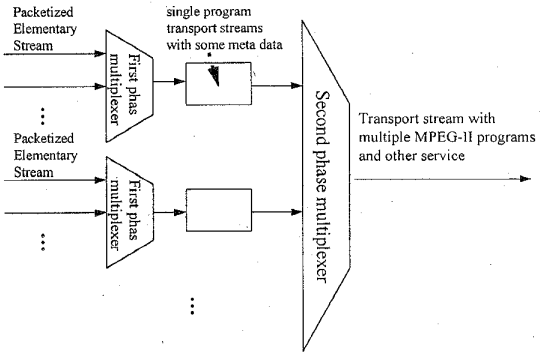

Fig. 2. Two-phase multiplexer [12]

3. Scheduling algorithms

We now present two types of scheduling algorithms : timestamp-insensitive and timestamp-sensitive. The former does not look into the timestamps in TS packets, while the latter does.

A. Timestamp-insensitive CBR scheduling algorithm

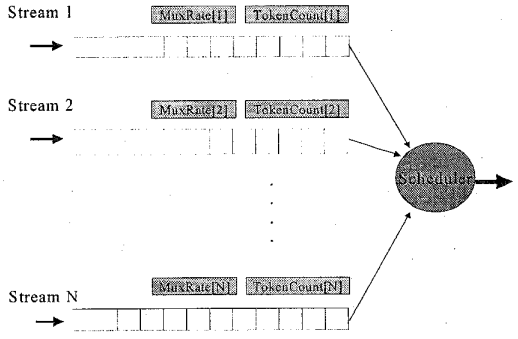

Fig. 3. The architecture of the CBR scheduling algorithm

In the VOD systems, the timestamp-insensitive CBR scheduling algorithm may be adopted to multiplex MPEG-II variable-bit-rate (VBR) video programs to simplify bandwidth allocation for each input stream. Fig. 3 shows the architecture of the CBR scheduling algorithm. Two variables *MuxRate*[*i*] and *TokenCount*[*i*] are maintained for each input stream *i*. The scheduling algorithm consists of two parts which are *Initialization* and *Scheduler*.

Initialization

Before a transport stream *i* enters the multiplexer, the second phase multiplexer needs to initialize *MuxRate*[*i*], $i = 1..N$, where N is the number of streams. The *MuxRate* is the meta data from the first phase multiplexer. The counter *TokenCount*[*i*] is then initialized as *MuxRate*[*i*], $i = 1..N$. Note that *MuxRate*[*i*] is a constant bit rate allocated to stream *i*, $i = 1..N$.

Scheduler

1. ***Initialization*:**

   Set I = { 1,2, .. ,N } ;
   For each new input stream $i \in I$
   {
   Set MuxRate[*i*] to the value from the first phase multiplexer ;
   TokenCount[*i*] = MuxRate[*i*] ;
   }

2. ***Scheduler*:**

   Find J = { *j* } such that TokenCount[*j*] = $\max_{i \in I}$ (TokenCount[*i*]) ;
   For $j \in J$
   {
   TokenCount[*j*] = TokenCount[*j*] – 1 ;
   Transmit one TS packet for stream *j* ;
   If (TokenCount[*i*] < 1) for all $i \in I$
   TokenCount[*i*] = TokenCount[*i*] + MuxRate[*i*] ;
   }
   Go back to the beginning of step 2 ;

Fig. 4. Timestamp-insensitive CBR scheduling algorithm

Fig. 4 is the timestamp-insensitive CBR scheduling algorithm. After initializing *MuxRate*[*i*] and *TokenCount*[*i*], $i = 1..N$, in each iteration, we find {*j*} such that *TokenCount*[*j*] is the maximum among all *TokenCount*[*i*], $i = 1..N$. Note that there might be multiple counters with the same values. For each stream *j*, *TokenCount*[*j*] is deducted by one and the multiplexer transmits one TS packet. If now all the counters have values smaller than one, the counters are wrapped back by adding their *MuxRate* respectively.

Token-Count	Program 1	Program 2	Program 3
	1.5 Mbps	3 Mbps	6Mbps
First cycle	1.5	3	6
	1.5	3	5

	1.5	3	4
	1.5	3	3
	1.5	2	2
	1.5	1	1
	0.5	1	1
	0.5	0	0
Second cycle	2	3	6
	2	3	5
	2	3	4
	2	3	3
	2	2	2
	1	1	1
	...	...	...

Fig. 5. An example run of the CBR scheduling algorithm [12]

An example run is given in Fig. 5 to illustrate how the multiplexing method works. We assume that the output capacity of the second phase multiplexer is 10.5 Mbps and the number of transport streams is 3. If the output capacity is larger than 10.5, the residual capacity will be used to transmit dummy TS packets in each iteration.

B. Timestamp-sensitive scheduling algorithm

This scheduling algorithm dynamically allocates bandwidth to each input stream according to the statistics collected in real time such that all the input streams can share the whole capacity of the output link. Bandwidth sharing can be explained by the StatMux of Imedia [13] in Fig. 6.

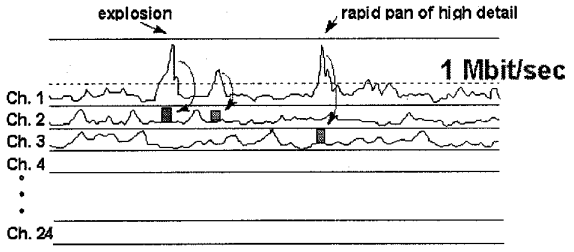

Fig. 6. An example of bandwidth sharing [Source : StatMux of IMedia] [13]

Because the MPEG-II encoder compresses a video program at a constant frame rate, e.g. 25 frames/sec, and outputs a coded stream with a highly variable instantaneous bit rate. The coded VBR stream then is multiplexed by the first phase multiplexer. The processing time of the first phase multiplexer is nearly constant for each input frame and the amount of header inserted into each input coded frame is proportional to the size of each frame. As a result, the output transport stream, which is also the input streams of our algorithm, of the first phase multiplexer is also variable-bit-rate with a constant frame rate and each frame consists of variable TS packets.

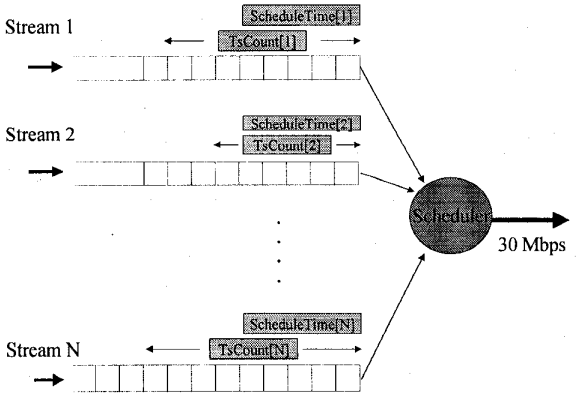

Fig. 7. The model of multiplexer with timestamp-sensitive scheduling algorithm

Fig. 7 shows the model of the multiplexer with our timestamp-sensitive scheduling algorithm listed in Fig. 8. The algorithm has two variables, ScheduleTime[*i*] and TsCount[*i*] :

ScheduleTime[*i*] stores the end time of current scheduling cycle and is initialized to be the initial time of input stream *i* plus a cycle time and increased by a cycle time at the beginning of each scheduling cycle. Note that the cycle time equals the frame period which is 0.04 sec, i.e. 1/25 sec, in our study. It is compared with the DTS of each input TS packet to decide whether the TS packet should be transmitted in current scheduling cycle to avoid deadline violation.

TsCount[*i*] stores the number of TS packets that should be transmitted in current scheduling cycle. The variable is increased by one when an incoming TS packet needs to be transmitted in current scheduling cycle, and it is decreased by one when the scheduler transmits a TS packet from stream *i*.

The scheduling algorithm consists of three parts, i.e. *Initialization*(), *DTS_check*(), and *Scheduler*(). *Initialization*() runs before a new input stream is served by the multiplexer. *Scheduler*() runs at the beginning of each scheduling cycle. *DTS_check*() is invoked every time a TS packet arrives to the multiplexer.

DTS_check() is used to update *TsCount*[*i*]. For stream *i*, when a TS packet arrives, *DTS_check*() checks the DTS of the TS packet. If the DTS is smaller than or equal to *ScheduleTime*[*i*], *TsCount*[*i*] is increased by one. Otherwise, *TsCount*[*i*] remains unchanged.

Scheduler() is responsible of allocating bandwidth to each stream. At the beginning of each scheduling cycle, it calculates the total urgent bandwidth demand,

$\sum_{i=1}^{N} TsCount[i]$. Then for each stream i, $TsCount[i]$ is copied to a temporary working variable $Count[i]$, and then reset to zero; meanwhile, $ScheduleTime[i]$ is increased by a cycle time. If the total urgent bandwidth demand exceeds the channel capacity, *MAXTS*, deadline violation will occur and *Scheduler*() runs iteratively to schedule and transmit TS packets. In each iteration, it first finds the set $S = \{i\}$ where stream i has non-zero $Count[i]$ and calculates the average residual bandwidth, *AvgTs*, which equals to the total residual bandwidth, i.e. *ResTs*, divided by the size, n, of S. Then for each stream i, if $Count[i]$ is more than *AvgTs*, the multiplexer transmits *AvgTs* TS packets and $Count[i]$ is deducted by *AvgTs*; otherwise, the multiplexer transmits only $Count[i]$ TS packets and $Count[i]$ is deducted to zero. In the latter case, the residual bandwidth from stream i, $AvgTs - Count[i]$, is added back to *ResTs* for scheduling in the next iteration. This process runs until *ResTs* equals to zero. The residual value in $Count[i]$ for each stream i is then added back to $TsCount[i]$. However, if the total urgent bandwidth demand is less than or equal to the channel capacity, the number of TS packets each stream can transmit is not only $Count[i]$ but also extra *Bonus*, which equals the total residual bandwidth, *ResTs*, divided by the number of streams, N. In that case, $TsCount[i]$, for stream i, is updated accordingly to reflect the extra packets being sent ahead of their schedules in this cycle.

```
Initialization()
{
    For each new input stream i
    {
        ScheduleTime[i] = Initial timestamp of stream i +
                          cycle time ;
        TsCount[i] = 0 ;
    }
}

DTS_check()
{
    Upon arrival of a TS packet of stream i, i = 1,..,N
    {
        Check DTS of the incoming TS packet;
        If ( DTS <= ScheduleTime[i] ) then
          TsCount[i] ++ ;
        If ( DTS > ScheduleTime[i] ) then
          TsCount[i] remains unchanged ;
    }
}

Scheduler()
{
    CycleTime = frame period ;
    MAXTS = (N * average bit rate of the input program * CycleTime) / (8*188) ;
    Count[i] = TsCount[i], i = 1,..,N ;
    TsCount[i] = 0, i = 1,..,N ;
    ScheduleTime[i] = ScheduleTime[i] + CycleTime, i = 1,..,N ;
    If ( sum_{i=1}^{N} Count[i] > MAXTS )
    {
        1. n = N ;
           Set T = S = { 1, 2, .. , N } ;
           ResTs = MAXTS ;
        2. AvgTs = ResTs / n ;
           For each stream i, i = 1,..,N
           {
             Transmit Min( Count[i], AvgTs ) TS
                packets for stream i ;
             If( Count[i] < AvgTs )
             {
               S = S - {i} ;
               n = n-1 ;
             }
           }
           ResTs = sum_{i ∈ T-S} (AvgTs - Count[i]) ;
           For each stream i, i = 1,..,N
             Count[i] = Count[i] – Min( Count[i],
                                        AvgTs ) ;
        3. If ( ResTs ≠ 0 )
             Goto step 2 ;
           else
             For each stream i, i = 1,..,N
               TsCount[i] = TsCount[i] + Count[i] ;
    }
    Else
    {
        1. ResTs = MAXTS ;
        2. Transmit Count[i] TS packets for stream i ;
           ResTs = ResTs - sum_{i=1}^{N} Count[i] ;
           Bonus = ResTs / N ;
           Transmit Bonus TS packets or dummy packets
             for stream i, i = 1,..,N ;
           TsCount[i] = TsCount[i] – Min(Bonus,
             TsCount[i]), i = 1,..,N ;
    }
}
```

Fig. 8. The timestamp-sensitive scheduling algorithm

Fig. 9 gives an example run of *Scheduler*(), where the total urgent demand exceeds the channel capacity. We assume that the number of streams is 4, and *TsCount*s of these four streams are 40, 80, 75, 55, respectively. The channel capacity, in units of TS packets per cycle, is 240. Note that in this scheduling cycle, program 2 has 10 TS packets which suffer deadline violation.

Count	Program 1	Program 2	Program 3	Program 4	ResTs	n	AvgTs
	35	90	70	55	240	4	60
1st iteration	0	30	10	0	25+5	2	15
2nd iteration	0	15	0	0	5	1	5
Final iteration	0	10	0	0	0	1	0

Fig. 9. An example run of Scheduler(), where the total urgent demand exceeds the channel capacity

4. Numerical Results

A trace-driven simulation experiment was conducted to study the performance of the second phase multiplexer using our timestamp-sensitive scheduling algorithm. The input streams to the second phase multiplexer are randomly captured from the Disney carton "The Lion King", with length of forty-eight seconds long for each stream. We assume that the packet-per-frame sequence of the original program is a stationary stochastic process. As a result, the randomly captured streams are independent and can be treated as streams of different programs.

Three measures are calculated as the simulation results. These three measures are standard deviation of the number of output data bits per frame period which is 0.04 sec, in each stream, the deadline violation probability, and the standard deviation of the number of delayed frames which represents the degree of jitter. Besides, the performance of another scheduling algorithm, i.e. the timestamp-insensitive CBR scheduling algorithm, is also measured and compared with our algorithm. The CBR scheduling algorithm transmits every input stream at a constant bit rate. The input streams to these two algorithms are the same.

In VOD systems over cable TV networks, an analog channel, capable of carrying 30 Mbps digital data, can carry 9 programs with an average bit rate of 3.26 Mbps. In our simulation model, we assume the capacity of the output link to be 9*3.26 Mbps.

A. Smoothing performance on 9 input streams using our timestamp-sensitive scheduling algorithm

The standard deviations of the numbers of *input* and *output* data bits per frame period in the original and smoothed streams, respectively, are shown in Table 1. Fig. 10 shows, for a stream selected from 9 streams, the number of TS packets per frame period for its input and output streams. The smoothing performance of the other 8 streams are similar. The original length of the input stream is about 1200 frames but only the first 120 frames are shown in Fig. 10. The result behind 120th frame are not shown because the output sequence keeps nearly constant at 86 TS packets per frame period from the 88th period to the last.

Stream Number	1	2	3	4	5
Original Stream	41.27	53.73	43.33	55.37	38.92
Smoothed Stream	4.14	10.14	7.82	9.45	7.40

Stream Number	6	7	8	9	Average
Original Stream	47.84	45.21	44.87	50.61	46.79
Smoothed Stream	6.85	7.18	8.20	8.71	7.77

Table 1. Standard deviation of the number of data bits in the original and smoothed streams (unit : TS packet)

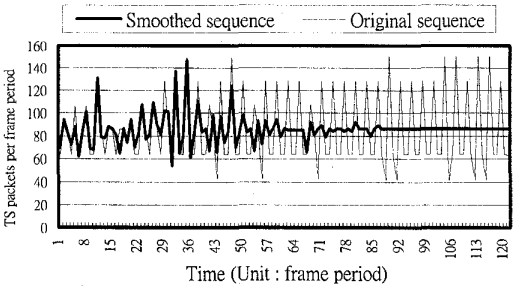

Fig. 10. The original sequence and the smoothed sequence for one of the 9 input streams

As shown in Table 1, the average standard deviation of the bursty input streams is greatly reduced from 46.79 to 7.77 by our timestamp-sensitive scheduling algorithm. The standard deviation from the 1st to the 88th frame periods is also reduced from 27.41 to 15.93.

B. Comparison of delay performance between CBR scheduling algorithm and timestamp-sensitive scheduling algorithm

Stream Index	1	2	3	4	5
Timestamp-insensitive CBR Schedule	0.1444	0.7007	0.0268	0.1792	0.0219

Timestamp-sensitive Schedule	0.0008	0.0008	0.0017	0.0017	0.0017
Standard Deviation of bit rate	41.27	53.73	43.33	55.37	38.92

Stream Index	6	7	8	9	Average
Timestamp-insensitive CBR Schedule	0.2636	0.1427	0.0211	0.0073	0.1675
Timestamp-sensitive Schedule	0	0.0008	0.0008	0.0008	0.0010
Standard Deviation of bit rate	47.84	45.21	44.87	50.61	46.79

Table 2. Deadline violation probabilities of two scheduling algorithms

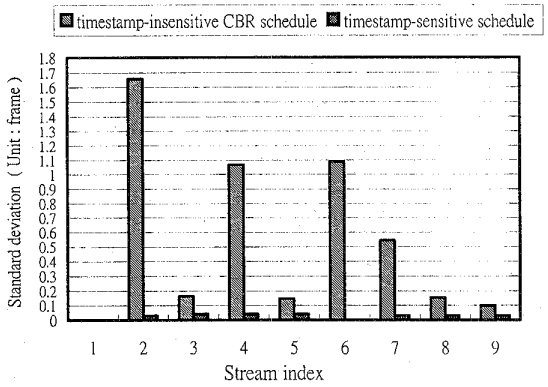

Fig. 11. Comparison of standard deviation of the number of delayed frames

Table 2 and Fig. 11 compare the deadline violation probability and the standard deviation of the number of delayed frames in the CBR and timestamp-sensitive scheduling algorithms. The deadline violation probability is calculated by dividing the number of delayed frames by the number of all frames. Our timestamp-sensitive algorithm achieves much reduction of deadline violation probability and standard deviation.

C. Improvement on the timestamp-sensitive scheduling algorithm

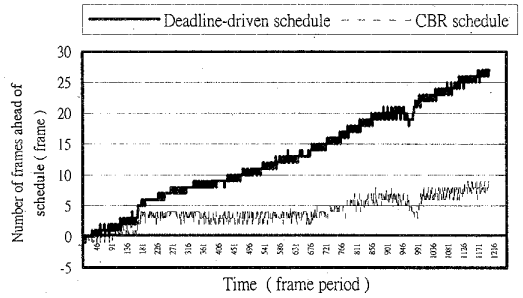

Fig. 12. The number of frames which are ahead of schedule in timestamp-sensitive and CBR scheduling algorithms

In our timestamp-sensitive scheduling algorithm, if the total bandwidth demand is smaller than or equal to the capacity of the output link, the scheduler will over-allocate the remaining bandwidth, which is called *Bonus*, to input streams. No limitation on *Bonus* results in unlimited over-allocation to each input stream. Fig. 12 shows the number of frames which are ahead of schedule, i.e. the decoding timestamp of a frame is larger than the time at which the last TS packet of the frame arrives at the decoder. The multiplexer or the set-top box (STB) might be overflowed. As a result, we limit the amount of *Bonus* to reduce the buffer requirement in the multiplexer and the STB.

Once the *Bonus* is limited, the effect of smoothing bursty programs decreases. This also results in the increase in jitter and deadline violation probability. However, we explore the relation between the value of *Bonus* and the degree of jitter and deadline violation probability. Then we find an optimal *Bonus* value to minimize the degree of jitter and deadline violation probability while avoiding buffer overflow.

As shown in Fig. 13 and Fig. 14, the average standard deviation and average deadline violation probability of the 9 output streams fall down rapidly in the case where the *Bonus* is limited to 1 and remain stable at 0.03 for the standard deviation and 0.001 for the deadline violation probability in the cases where the *Bonus* is equal to or more than 2.

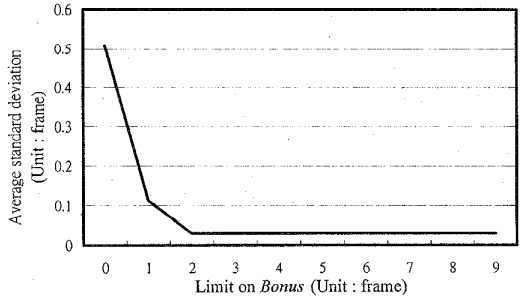

Fig. 13. The average standard deviation of the number of delayed frames of 9 output streams as a function of *Bonus* limitation

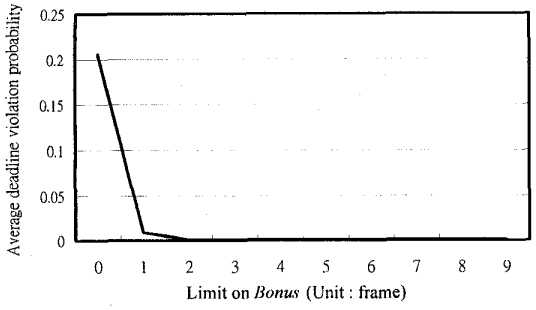

Fig. 14. The average deadline violation probability of 9 output streams as a function of *Bonus* limitation

D. Comparison of delay performance between three scheduling algorithms

Stream Index	1	2	3	4	5
CBR	0.1444	0.7007	0.0268	0.1792	0.0219
Bonus limitation = 2	0.0008	0.0008	0.0016	0.0016	0.0016
Bonus Unlimited	0.0008	0.0008	0.0016	0.0016	0.0016
Standard deviation of bit rate	41.27	53.73	43.33	55.37	38.92

Stream Index	6	7	8	9	Average
CBR	0.2636	0.1427	0.0211	0.0073	0.1675
Bonus limitation = 2	0	0.0008	0.0008	0.0008	0.0010
Bonus Unlimited	0	0.0008	0.0008	0.0008	0.0010
Standard deviation of bit rate	47.84	45.21	44.87	50.61	46.79

Table 3. Comparison of deadline violation probability for three scheduling algorithms

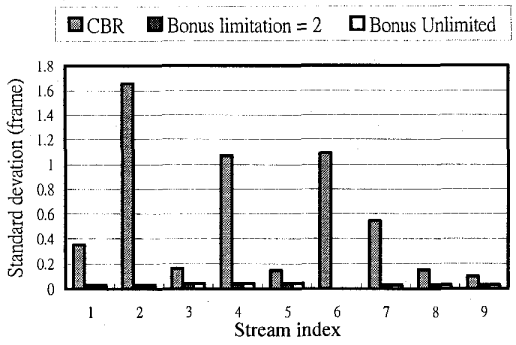

Fig. 15. Comparison of standard deviation of the number of delayed frames for three scheduling algorithms

Table 3 compares the deadline violation probability of three scheduling algorithms, i.e. the timestamp-insensitive CBR scheduling algorithm, the timestamp-sensitive scheduling algorithm with *Bonus* limited to 2, and the timestamp-sensitive scheduling algorithm without limitation on *Bonus*. Fig. 15 compares the standard deviation of the number of delayed frames for these three algorithms. From Table 3 and Fig. 15, it is demonstrated that the performance of timestamp-sensitive scheduling algorithm with *Bonus* limited to 2 is much better than the CBR scheduling algorithm. The timestamp-sensitive scheduling algorithms with *Bonus* limited to 2 and without *Bonus* limitation have the same performance in jitter and deadline violation probability. However, the former can prevent buffer overflow.

The multiplexer can keep 2 frames for each stream beforehand and then starts to schedule the input streams, while each STB can allocate a buffer of 2 frames for its input stream. This will prevent the multiplexer queues from underflow and the STB from overflow.

5. Conclusion

This paper presents a timestamp-sensitive scheduling algorithm for MPEG-II multiplexers. The algorithm combines the virtues of statistical multiplexing and smoothing. It takes into account the decoding timestamp of transport stream packets in scheduling variable-bit-rate streams and avoids deadline violation at the set-top-boxes.

Results of a trace-driven simulation demonstrate that the deadline violation probability and delay standard deviation obtained using our timestamp-sensitive scheduling algorithm are much smaller than those obtained using the timestamp-insensitive CBR scheduling algorithm. Besides, a limitation on the number of over-scheduled frames is imposed to prevent from overflowing the buffers in STBs and underflowing the buffers in multiplexers. A trace-driven simulation finds that the enhanced version of the timestamp-

sensitive scheduling can achieve the same delay performance as our original scheduling algorithm, but no buffer overflow, with only two frames stored beforehand in the buffer of the multiplexer and a buffer size of two frames at the STB.

Acknowledgement

The authors would like to thank Mentor Data System Corp. at Science-based Industrial Park, Hsinchu, Taiwan, for providing MPEG-II trace programs and technical advice on MPEG-II multiplexers.

Reference

[1] Yu-Lin Chang, "Protocol Design and Implementation of Video-On-Demand Service over CATV", Section 2.1, Master Thesis, Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan, June 1997.

[2] M. Riyaz Ismalil, I. E. Lambadaris, M. Devetsikiotis and A. R. Kaye, "Modelling prioritized MPEG video using TES and a frame spreading strategy for transmissiion in ATM networks", Proceedings. IEEE INFOCOM '95, Vol. 2, p.762-70.

[3] Simon S. Lam, Simon Chow, and David K. Y. Yau, "An algorithm for lossless smoothing of MPEG video", SIGCOMM '94.

[4] Mike Perkins and David Arnstein, "Statistical Multiplexing of Multiple MPEG-2 Video Programs in a Single Channel", Technical Paper, SMPTE Journal, September 1995.

[5] Ajanta Guha and Daniel J. Reininger, "Multichannel joint rate control of VBR MPEG encoded video for DBS applications", IEEE Transactions on Consumer Electronics, Vol. 40, Vol. 3, p.616-23, August 1994.

[6] C. Blondia and O. Casals, "Performance analysis of statistical multiplexing of VBR sources", INFOCOM '92, p.828-38.

[7] P. Pancha and M. El Zarki., "Bandwidth requirements of variable bit rate MPEG sources in ATM networks", In Proceedings of INFOCOM '93, pages 902-909, March 1993.

[8] L. Delgrossi, C. Halstrick, D. Hehmann, R. G. Herrtwich, O. Krone, J. Sandvoss, and C. Vogt., "Media scaling for audiovisual communication with the Heidelberg transport system", In Proceedings of ACM Multimedia '93, pages 99-104, August 1993.

[9] H. Kanakia, P. Mishra, and A. Reibman. "An adaptive congestion control scheme for real-time packet video transport", In Proceedings of ACM SIGCOMM '93, pages 20-31, September 1993.

[10] Draft International Standard ISO/IEC DIS 13818-2, "Information Technology – Generic Coding of Moving Pictures and Associated Audio – Part 2 : Video", May 1994.

[11] ITU-T Rec. H.222.0/ISO/IEC 13818-1 International Standard, "Information Technology – Generic Coding of Moving Pictures and Associated Audio : Systems", ISO/IEC JTC1/SC29/WG11 N0801rev., Nov. 1994.

[12] Jau-Ming Jan,"Design and Implementation of MPEG-II MUX/DEMUX and Transport Protocol over CATV Networks", Master Thesis, Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan, June 1996.

[13] Imedia corporation, "24 Digital TV Channels in the Space of a Single Analog Channel", http://www.imedia.com

Biography

Ying-Dar Lin received the Bachelor's degree in Computer Science and Information Engineering from National Taiwan University in 1988, and the M.S. and Ph.D. degrees in Computer Science from the University of California, Los Angeles in 1990 and 1993, respectively. He joined the faculty of the Department of Computer and Information Science at National Chiao Tung University in August 1993 and is now Associate Professor. His research interests include design and analysis of high-speed LANs/MANs/WANs, traffic characterization, service and network management, and network-centric computing. Dr. Lin is a member of ACM and IEEE. He can be reached at ydlin@cis.nctu.edu.tw.

Chun-Mo Liu received his B.S. and M.S. degrees in Computer and Information Science at National Chiao Tung University, Hsinchu, Taiwan, in 1996 and 1998, respectively. His thesis work focuses on multiplexing MPEG-II program streams for statistical gain and

Internet access over cable networks. He is now on his two-year duty in the air force of Taiwan.