Introduction

Since video applications typically consume large network bandwidth, video data needs to be compressed before its transmission over a network. Many coding schemes have been proposed/developed for *variable bit rate* (VBR) video services; most notable among these is the coding scheme developed by the Moving Pictures Experts Group (MPEG). MPEG's original design goal was to develop an encoding scheme for storing video (together with the associated audio) on digital storage media. MPEG compression has also been proven to be suitable for delivering video frames over networks.

The MPEG standard [7] includes three types of encoded frames: I (intrapictures), P (predicted pictures), and B (interpolated pictures/bidirectional prediction). I-frames exploit the spatial redundancy within a frame and are coded independently of other frames. P- and B-frames exploit the temporal redundancy present in a video sequence (stream) and are coded with reference to other P- and/or I-frames. P-frames update the picture (using a predictive algorithm) from the last I- or P-frame. B-frames use the bidirectional prediction method and are coded with reference to both the past and the future I- or P-frames. In general, I-frames are much larger than P-frames, and P-frames are much larger than B-frames. Two parameters, M and N, are used to specify a sequence of encoded pictures, where M is the interval between an I-frame and a P-frame or two P-frames, and N is the interval between two I-frames. An example of an MPEG video stream with $M = 3$ and $N = 9$ is shown in Fig. 1.

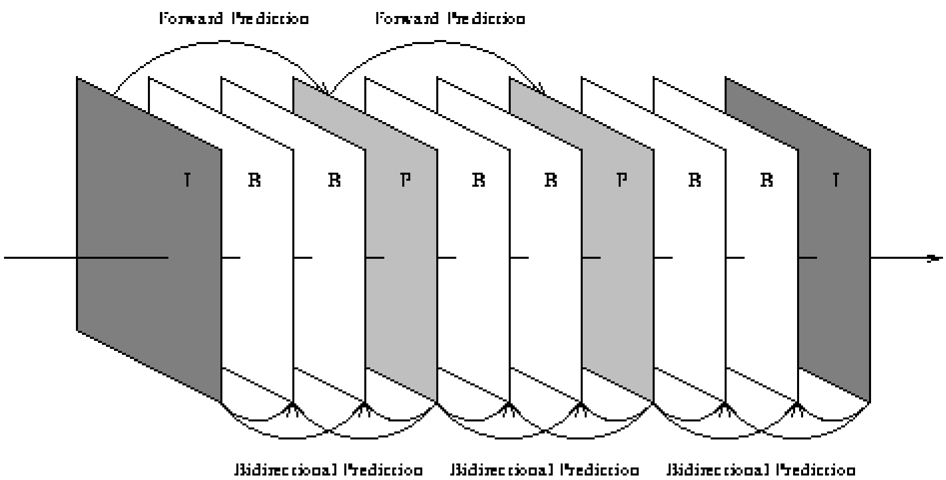

Figure 1: An MPEG-compressed video stream with $M = 3$ and $N = 9$.

Since I-frames are coded independently of other frames, they prevent the propagation of coding errors. I-frames are therefore more important than P- or B-frames in reconstructing pictures, and the transmission of I-frames should have a higher priority than that of P- or B-frames. Also, in order to provide high-quality continuous video, frames must be displayed/played back at a rate of 30 frames/second, and loss of a frame is usually better than displaying it at a wrong place/time. Thus, a late frame should be replaced by an estimated frame (from a previous frame) and displayed at the right time. A good transmission scheduling scheme should therefore adopt a *better-never-than-late* policy in transmitting video frames; that is, if a frame cannot be delivered before its deadline, the scheduling scheme should simply discard the frame without wasting the server time to transmit a useless (late) frame. However, we should not discard too many (consecutive) frames since this can seriously degrade the quality of the reconstructed pictures at the receiver/client site.

As described in [2], the applications of digital video compression can be classified as *asymmetric* and *symmetric*. Asymmetric applications require frequent use of the decompression process, but use the compression process once and for all at the production of the video. For example, in case of movie delivery, each movie is encoded and stored in a digital storage medium. It is transmitted to, and decompressed and played back at, a remote client/receiver site upon request. Other examples of asymmetric applications include video-on-demand, electronic publishing, video games, etc. For asymmetric applications, since the frames are encoded and stored before their usage, it is reasonable to assume that the sizes of frames are known before the frames are scheduled for transmission.

Symmetric applications require essentially equal use of compression and decompression processes. For example, in case of video conferencing, video information is generated by a camera, compressed by an encoding scheme, transmitted over the network, and finally, decompressed and displayed at a remote client/receiver site. Other examples of symmetric applications include videophone, video mail, and desktop video publishing. In general, for symmetric applications, only the sizes of the frames that have been encoded are known to the server. If the server does not have a large buffer or if the frames are required to be displayed in real-time (soon after they are produced) at the client site, the server may have knowledge on the size of only one frame just before its transmission. That is, the server knows only the size of the frame that has been encoded and is currently waiting for transmission.

VBR video encoding schemes and their applications to video transmissions have been studied extensively. For example, Ott *et al.* [8] and Lam *et al.* [5] proposed smoothing schemes for VBR video. Reibman and Berger [12] and Reininger *et al.* [13] studied the problem of transporting/multiplexing VBR/MPEG video over ATM networks. Pancha and Zarki studied the MPEG video coding standard for the transmission of VBR video [9] and the performance of variable bandwidth allocation schemes for VBR MPEG video [10]. In this paper, we formulate the problem of scheduling the transmission of MPEG-compressed video streams with firm deadline constraints. We assume that each video stream is allowed to miss its frame deadline once in a while without seriously degrading the quality of the reconstructed pictures at the receiver site. Specifically, each video stream is allowed to miss at most one frame deadline in any K consecutive frames of the video stream, where K is a user- or application-specified parameter. However, as mentioned earlier, since I-frames are coded independently of other frames and can prevent the coding error propagation, we require *all* I-frames to meet their transmission deadlines. We propose a transmission scheduling scheme for a set of MPEG-compressed video streams and discuss the effectiveness of the proposed scheme.

The rest of the paper is organized as follows. The problem of scheduling MPEG-compressed video with firm deadline constraints is formally stated in Section 2. Section 3 describes the proposed transmission scheduling scheme. A few important remarks on the proposed scheme are given in Section 4. The paper concludes with Section 5.

Table of Contents

Problem Formulation

Consider a system with a single server and n independent streams $\mathbf{S}$ = $\{S_i = (P_i, K_i, Q_i) \mid i = 1, 2, \ldots, n\}$ of MPEG-coded video frames as shown in Fig. 2. Each video stream S_i is characterized by three parameters P_i, K_i, and Q_i, where P_i is the transmission period of S_i (i.e., one frame from S_i is to be transmitted in each time interval of length P_i), K_i is the minimum tolerable interval between two frame deadline misses (i.e., at most one frame in any window of K_i frames is allowed to miss its transmission deadline without considerably degrading the quality of reconstructed pictures at the receiver), and Q_i is the number of frames in S_i whose sizes are known *a priori*. The frames in each video stream are numbered and transmitted in playback order. Note, however, that to decompress and display/play back a B-frame at the receiver site, the corresponding referenced I- and/or P-frames must also be present at the receiver site. Therefore, we assume that at all times at least M frames are transmitted to the receiver site ahead of time and the buffer size at the receiver site is large enough to store these frames, where M is the interval between an I-frame and a P-frame or two P-frames. That is, when the j-th frame is to be decompressed and displayed at the receiver site, all the ℓ-th frames, $j \leq \ell \leq j + M - 1$, have already been transmitted to and stored at the receiver site.

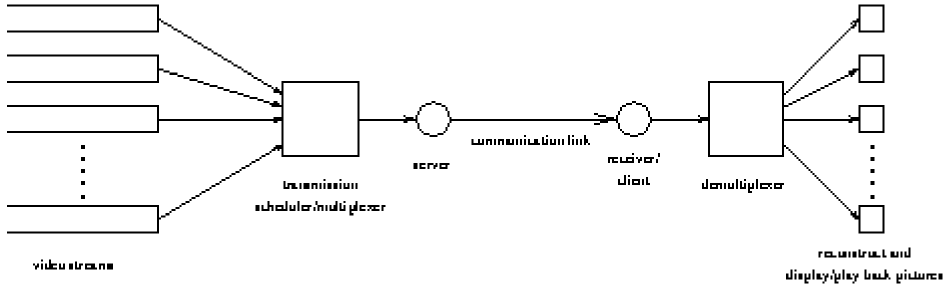

Figure 2: System model.

More specifically, let F_{ij} denote the j-th frame in video stream S_i, for $1 \le i \le n$ and $j \ge 1$. If the first frame F_{i1} in S_i is ready for transmission at time r_{i1}, then the j-th frame F_{ij} in S_i is ready for transmission at time $r_{ij} = r_{i1} + (j-1) \cdot P_i$ and must complete its transmission by time $d_{ij} = r_{i1} + j \cdot P_i$. We call r_{ij} and d_{ij} the *ready time* and *deadline* of F_{ij}, respectively, for $1 \le i \le n$ and $j \ge 1$. As contemporary networks tend to use fixed-length packets, such as the 53-byte ATM (*Asynchronous Transfer Mode*) cells, we assume that frames are decomposed into fixed-length packets/cells before their transmission, and all timing parameters, such as periods, ready times, and deadlines, are specified/measured in *slots*, where a slot is the packet- or cell-transmission time. (We will henceforth use the term ``cell.'') Therefore, the units of time and frame size are slot and cell, respectively, and the time interval $[t-1, t)$ is called *slot* t.

Let C_{ij} denote the size of frame F_{ij} (measured in cells). The remaining (yet-to-be-transmitted) part of F_{ij} decreases with the progress in transmitting F_{ij}. To reflect this, we use $C_{ij}(t)$ to denote the remaining frame size of F_{ij} at time t for $r_{ij} \le t \le d_{ij}$. Note that $C_{ij}(t) = C_{ij} \; \forall \; t \le r_{ij}$ and $C_{ij}(d_{ij}) > 0$ implies that F_{ij} missed its deadline. Because it is useless to transmit a video frame after its deadline is expired, we define $C_{ij}(t) = 0 \; \forall \; t > d_{ij}$. Moreover, if at a certain time t we decide to discard frame F_{ij} without transmitting it at all or abort its in-progress transmission (because F_{ij} is found unable to meet its deadline after all), we define $C_{ij}(s) = 0 \; \forall \; s > t$. A discarded/aborted frame is also thought of as missing its deadline.

A frame F_{ij} is called the *current frame* of S_i at time t if F_{ij} is ready at time t and its deadline has not yet been expired, i.e., $r_{ij} \le t < d_{ij}$. Note that at any time $t \ge r_{i1}$ and before the termination of the transmission of stream S_i, there is exactly one current frame in S_i for all i. If F_{ij} is the current frame of S_i at time t, then $F_{ix}, x < j$ ($x > j$), is called a *past* (*future* or *next*) frame of S_i at time t. Suppose F_{ij} is the current frame of S_i at time t. Then, the sizes $C_{i,j+x}(t)$ of frames $F_{i,j+x}, 0 \le x < Q_i$, in S_i are known to the server (scheduler) at time t. As mentioned earlier, the value of Q_i depends on the underlying application. For asymmetric digital video applications, Q_i is usually very large (or virtually infinite) since all the video frames are encoded and stored in a digital storage medium before their transmission starts. On the other hand, for symmetric digital video applications, Q_i is usually small since each picture is captured, encoded, and transmitted in real-time. Note, however, that at least the size of the current frame in a video stream is known to the scheduler for both symmetric and asymmetric applications (i.e., $Q_i \ge 1$).

Since I-frames are more important in reconstructing pictures at the receiver site, they must be transmitted before their deadlines. Also, in order to avoid flickering, video streams are not allowed to miss their P- or B-frame deadlines too often. If an I-frame misses its deadline or if more than one frame in any window of K_i frames misses its deadline, a dynamic failure is said to have occurred. Our problem is then to multiplex/schedule the transmission of n video streams on the server so as to (i) minimize the probability of dynamic failure and (ii) transmit as many video frames before their deadlines as possible.

In our problem formulation, we assume that there are n video streams in the system, but we actually allow dynamic video connection and termination requests because each $r_{i1}, 1 \le i \le n$, can be an arbitrary number. That is, we allow a new video stream S_ℓ to be requested at any time $r_{\ell 1}$, and an existing video stream S_i to be terminated at any time $d_{ij} > r_{i1}$ for some $j \ge 1$. It is also worth mentioning that although our problem formulation and proposed transmission scheduling scheme are motivated by MPEG-coded video transmission, they are applicable to the transmission of general (not necessarily MPEG-coded) video/message streams.

Our problem formulation is similar to that of the customer service problem with (M, K)*-firm deadlines* studied in [3]. A stream of customers/messages is said to have (M, K)-firm deadline if at least M customers in any window of K consecutive customers (from the stream) must meet their deadlines; otherwise, a dynamic failure is said to occur. It is easy to see that, in our problem formulation, in addition to the requirement that each I-frame must meet its deadline, each video stream S_i is also subject to, in their term, the $(K_i - 1, K_i)$-firm deadline constraint. In this sense, our problem is more restricted than theirs. However, our problem formulation is more suitable for video transmissions for the following reason. In their formulation, $K - M$ video frames from a stream are allowed to miss their deadlines in a window of K consecutive frames from the stream. Suppose $K - M > 1$. In their formulation, it is acceptable if $K - M$ consecutive frames miss their deadlines. However, missing consecutive frame deadlines may cause a serious degradation in the quality of reconstructed pictures. So, in most applications, even if we allow $K - M$ frames to miss their deadlines in a window of K consecutive frames, we usually want the frame deadline misses to be uniformly distributed among the K consecutive frames. Hence, we can transform the (M, K)-firm deadline constraint to a $(K' - 1, K')$-firm deadline constraint, where $K' = \lceil K/(K - M) \rceil$. It is easy to see that if a video stream meets the $(K' - 1, K')$-firm deadline constraint, then it also meets the (M, K)-firm deadline constraint (but not the converse).

The authors of [3] proposed a simple service policy for streams with (M, K)-firm deadlines. Their main idea is that each stream is assigned a priority equal to the minimum number of consecutive misses required to take the stream from its current state to a failing state, where a larger priority value means a lower priority. Then, the server always chooses to service next the stream with the highest priority among the streams with queued customers/frames. For streams of the same priority level, the order of service is determined according to the *earliest-deadline-first* (EDF) policy [6], i.e., the frame with the earliest deadline gets to be serviced first. Their approach is quite simple and intuitive. However, the main drawback of their approach is that it does not consider the timing constraints among streams of different priority levels. For example, a frame from a higher-priority stream may have a longer deadline than that from a lower-priority stream. If frames from a higher-priority stream are always serviced first, frames from a lower-priority stream may miss their deadlines. This may cause unnecessary deadline misses, which, in turn, result in raising the priority of the lower-priority stream, thereby making the entire system more difficult to schedule. However, if we also want to take the timing constraints into consideration, we may have to service the frames with earlier deadlines first as long as we can still guarantee that the frames from higher-priority streams meet their deadlines. The following example further illustrates this point.

Example 1: Let $\mathbf{S} = \{S_1, S_2\}$ and $(P_1, K_1, Q_1) = (6, 2, 1)$ and $(P_2, K_2, Q_2) = (9, 2, 1)$. Suppose S_1 is ready at time 1 and S_2 is ready at time 0, i.e., $r_{11} = 1$ and $r_{21} = 0$. Moreover, assume $C_{1j} = 3$ for all $j \geq 1$, and $C_{2j} = 8$ for all $j \geq 1$. The schedule produced by the service policy described in [3] is shown in Fig. 3 (a). F_{21} is first transmitted in time interval $[0, 1)$ (time slot 1), and then preempted by F_{11} at time 1 since F_{11} has an earlier deadline than F_{21} ($d_{11} = 7 < d_{21} = 9$). At time 4, F_{11} finishes its transmission, and the server switches back to transmit F_{21}. However, since F_{21} cannot meet its deadline, we discard/abort it, and raise S_2's priority to a level higher than S_1. At time 7, the server starts to transmit F_{12}, and then at time 9, F_{22} preempts F_{12} and the server transmits F_{22} from time 9 to time 17. Note that at time 13, F_{12} misses its deadline, and we raise S_1's priority to the same level as S_2. But, since F_{22} has an earlier deadline than F_{13} ($d_{22} = 18 < d_{13} = 19$), F_{22} will continue its transmission until time 17. Now, at time 17, F_{13} cannot meet its deadline, and a dynamic failure occurs. The same situation occurs from time 18 to time 37. It is easy to see that there is one dynamic failure and there are only two frames $F_{1,3j-2}$ and $F_{2,2j}$ meeting their deadlines in each time interval $[(j-1)P_2,\ 1 + 3j \cdot P_1) = [9j - 9,\ 18j + 1)$ for all $j \geq 1$.

In contrast, we will show that using our scheme, the schedule produced looks like the one shown in Fig. 3 (b). At time 9, because F_{21} misses its deadline, F_{22} becomes urgent (since if F_{22} also misses its deadline, a dynamic failure occurs). However, instead of raising S_2's (F_{22}'s) priority and preempting F_{12}, we pre-schedule F_{22} as late as possible (i.e., from time 10 to time 18), and then, continue the transmission of F_{12} (at time 9). At time 10, the transmission of F_{12} completes, and we start to transmit F_{22}. At time 13, F_{13} becomes ready. But since F_{22} is more urgent, we continue its transmission, and F_{13} misses its deadline. The same situation occurs from time 18 to time 37. It is easy to see that there is no dynamic failure and there are three frames $F_{1,3j-2}$, $F_{1,3j-1}$, and $F_{2,2j}$ meeting their deadlines in each time interval $[(j-1)P_2,\ 1+3j\cdot P_1) = [9j-9,\ 18j+1)$ for all $j \geq 1$.

However, if $C_{1,3j-2} = 3$, $C_{1,3j-1} = 4$, and $C_{1,3j} = 2$, using the service policy described in [3], the schedule produced looks like the one shown in Fig. 3 (c), in which there is no dynamic failure and there are three frames $F_{1,3j-2}$, $F_{1,3j}$, and $F_{2,2j}$ meeting their deadlines in each time interval $[(j-1)P_2, 1+3j\cdot P_1) = [9j-9, 18j+1)$, for all $j \geq 1$. In contrast, using our scheme, at time 9, we will pre-schedule F_{22} from time 10 to time 18. However, when we resume the transmission of F_{12} at time 9 after we have pre-scheduled F_{22}, we find that F_{12} cannot meet its deadline since the only time interval left for F_{12} before its deadline is $[9, 10)$. In this situation, our scheme will not transmit F_{12}; instead, it will transmit F_{22}. At time 13, since F_{12} misses its deadline, F_{13} becomes urgent. We pre-schedule F_{13} from time 17 to time 19, and reschedule F_{22} from time 13 to time 17 (note that its remaining frame size is now 4), and then resume the transmission of F_{22} (details of our scheme will be discussed in Section 3). As it will be clearer later, the schedule produced by our scheme is the same as that shown in Fig. 3 (c). □

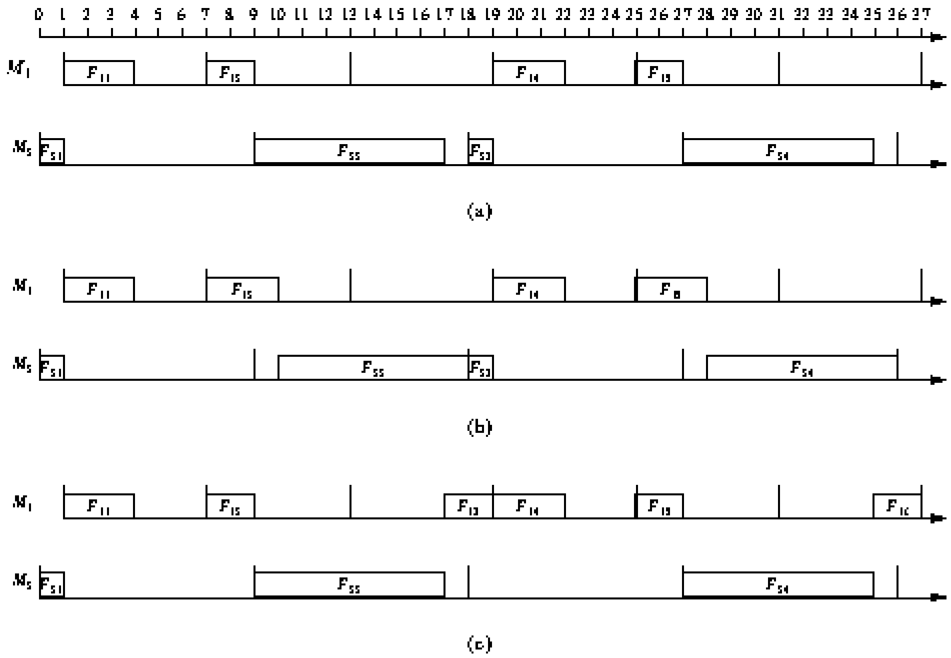

Figure 3: Schedules produced for the video streams described in Example 1.

The basic idea behind our approach is to consider, in addition to the urgencies, the deadlines of the frames in different video streams. When a frame in stream S_i misses its deadline, the next $K_i - 1$ frames in S_i become urgent (note that I-frames are always urgent). We want to guarantee the deadlines of urgent frames without sacrificing the chances of meeting the deadlines of non-urgent frames. In order to achieve this goal, we pre-allocate the server time to the urgent frames in the next H_i frames (including the current frame) of stream S_i as late as possible, and transmit non-urgent frames first, where H_i ($1 \leq H_i \leq Q_i$) is called the *lookahead number* of S_i. If we reach a point, called the *notification time* of an urgent frame - if the urgent frame does not start its transmission on or before this time then it will miss its deadline - then we preempt the current non-urgent transmission and send the urgent frame whose notification time has reached. However, if there is no non-urgent frame waiting for transmission and none of the notification times of the urgent frames have been reached, we choose an urgent frame to transmit next so that the server will not be left idle if there are frames ready for transmission. In the next section, we describe in detail the proposed scheduling scheme for transmitting MPEG-coded video streams.

Table of Contents

The Proposed Transmission Scheduling Scheme

A single server is responsible for handling a set $\mathbf{S} = \{S_i = (P_i, K_i, Q_i) \mid i = 1, 2, \ldots, n\}$ of video streams. A frame F_{ij} from stream S_i is said to be *active* at time t if (i) it is the current frame of S_i at time t, (ii) it is not discarded/aborted, and (iii) its transmission has not yet been completed (i.e., $r_{ij} \le t < d_{ij}$ and $C_{ij}(t) > 0$). A frame F_{ij} is said to be *urgent* if F_{ij} is an I-frame or if for some x such that $\max(1, j - K_i + 1) \le x < j$, frame F_{ix} does not (or is determined not to) meet its deadline. A frame is said to be *normal* if it is not urgent. A video stream S_i is in the *urgent state* at time t if at least one of the past $K_i - 1$ frames missed its deadline, i.e., if F_{ij} is the current frame of S_i at time t and for some x such that $\max(1, j - K_i + 1) \le x < j$, frame F_{ix} missed its deadline. A video stream is said to be in the *normal state* if it is not in the urgent state.

Using the EDF policy to transmit video frames, the server always chooses to transmit the frame with the earliest deadline among all active frames. Note that EDF is an on-line priority-driven preemptive [6] scheduling policy. For preemptive scheduling, the transmission of video frames can be suspended at any time and resumed at a later time. For priority-driven scheduling, each video frame is given a specific level of priority, and the server always chooses to transmit the highest-priority frame among all active frames. For EDF scheduling, each frame is assigned a priority according to its deadline. If two frames have the same deadline, the frame that the server chooses to transmit first (according to any tie breaking rule) is said to have a higher priority than the other.

To pre-schedule a set of urgent frames, we propose a pre-scheduling algorithm, called *backwards*-EDF, as follows. Given a set of frames, the backwards-EDF algorithm ``reverses'' the time axis and treats the ready time and deadline of a frame as its ``deadline'' and ``ready time,'' respectively, and pre-schedules these frames according to the backwards-EDF rule. Using the backwards-EDF scheduling algorithm, a frame F_{ij} is said to have a higher priority than another frame F_{xy} if the ready time r_{ij} of F_{ij} is larger than the ready time r_{xy} of F_{xy}. If two frames have the same ready time, the frame that will be chosen by the server to pre-schedule first (according to any tie breaking rule) is said to have a higher priority than the other. Note that backwards-EDF is *not* an on-line scheduler; it is a pre-scheduler used to pre-allocate the server to a (finite) set of video frames. It is well-known that EDF is optimal in the sense that if a set of frames is schedulable by any other service policy, then it is also schedulable by EDF. Therefore, it is easy to see that the backwards-EDF algorithm is also optimal.

For example, given a set of urgent video frames $\{F_i \mid 1 \le i \le 6\}$, suppose (r_i, d_i, C_i) = (14, 18, 2), (18, 22, 2), (26, 30, 2), (10, 15, 2), (25, 30, 2), and (11, 19, 3), where r_i, d_i, and C_i are the ready time, deadline, and frame size of F_i, for $i = 1, 2, \ldots, 6$, respectively. The allocation schedule produced by the backwards-EDF algorithm is shown in Fig. 4, where v_i is the start time of (the first reserved interval) of F_i, for $1 \le i \le 6$, respectively. Note that since $r_3 > r_5$, F_3 has a higher priority than F_5. Therefore, F_3 is pre-scheduled (backwards) from time 30 to time 28, and then F_5 is pre-scheduled from time 28 to time 26. Also note that, since F_6 has a lower priority than F_1 ($r_6 < r_1$), F_6 is preempted by F_1 from time 16 to time 18.

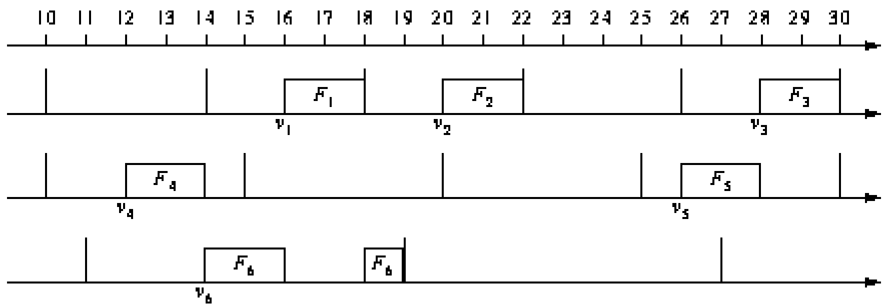

Figure 4: An example of a schedule produced by the backwards-EDF algorithm.

Our scheme uses an approach similar to the *last-chance* philosophy [1][4]: if there are normal frames waiting for transmission, an urgent frame will not be transmitted until the latest possible time (the *notification time*). In what follows, we describe the proposed transmission scheduling scheme and give illustrative examples.

The proposed scheme

Given a set of video streams $\mathbf{S} = \{S_i = (P_i, K_i, Q_i) \mid i = 1, 2, \ldots, n\}$, we use the EDF service policy to transmit the normal frames. Specifically, at a certain time t which was not reserved for any urgent frame, if there are normal frames waiting for transmission, the server chooses the active normal frame, say F_{ij}, with the earliest deadline d_{ij}. However, before the transmission of F_{ij}, the server first checks if F_{ij} can meet its deadline, i.e., check the inequality $C_{ij}(t) \le d_{ij} - t - R(t, d_{ij})$, where $R(t, d_{ij})$ is the number of slots that have been reserved at time t for urgent frames from time t to time d_{ij}. If F_{ij} cannot meet its deadline, we simply discard or abort F_{ij}. After F_{ij} is discarded/aborted, the next $K_i - 1$ frames $F_{i,j+1}, F_{i,j+2}, \ldots, F_{i,j+K_i-1}$ become urgent and video stream S_i enters the urgent state. We then pre-schedule the urgent frames of the next $H_i - 1$ frames $F_{i,j+1}, F_{i,j+2}, \ldots, F_{i,j+H_i-1}$ that have not been scheduled before. (Note that if $K_i < H_i$, only the next $K_i - 1$ frames $F_{i,j+1}, F_{i,j+2}, \ldots, F_{i,j+K_i-1}$ *and* I-frames are urgent.) We may also need to reschedule those pre-scheduled urgent frames whose pre-allocated transmission intervals are affected by the newly-scheduled urgent frames. We pre-schedule/reschedule these urgent frames as late as possible using the backwards-EDF algorithm, and record/update the notification time v_{xy} (of the first pre-allocated transmission interval) of each pre-scheduled/rescheduled frame F_{xy}. Note that the newly-scheduled urgent frames may affect only the pre-scheduled frames whose ready and current notification times are less than those of the newly pre-scheduled frames.

Example 2: Suppose at time t_0 frames F_1-F_4 are the urgent frames and have been pre-scheduled as shown in Fig. 5 (a). Moreover, assume F_u becomes urgent at time t_0 and needs to be pre-scheduled. Fig. 5 (b) shows the schedule after F_u is pre-scheduled using the backwards-EDF algorithm; note that F_1 and F_4 are rescheduled and their notification times v_1 and v_4 are changed accordingly. □

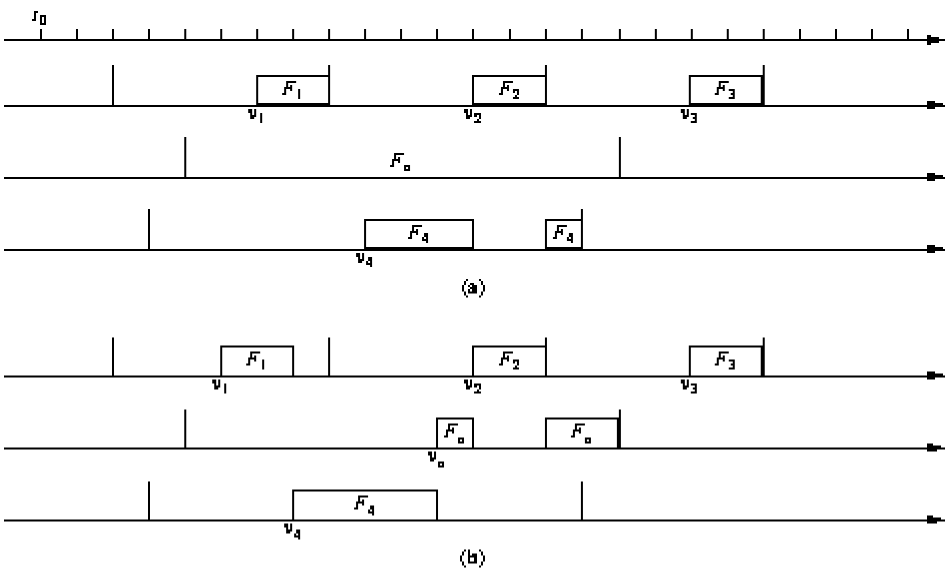

Figure 5: Pre-scheduling using the backwards-EDF algorithm.

Whenever the deadline of a frame F_{ij} is reached, $F_{i,j+H_i}$ is pre-scheduled using the backwards-EDF algorithm, and its notification time $v_{i,j+H_i}$ is recorded if $F_{i,j+H_i}$ is an urgent frame - that is, either $F_{i,j+H_i}$ is an I-frame or at least one of the frames $F_{i,j+H_i-1}, F_{i,j+H_i-2}, \ldots, F_{i,j+H_i-K_i+1}$ does not or will not meet its deadline. Also, those pre-scheduled urgent frames affected by this newly pre-scheduled frame $F_{i,j+H_i}$ will be rescheduled and their notification times will be updated. Note that after the pre-scheduling/rescheduling, if an urgent frame F_{xy} has a (new) notification time v_{xy} less than its ready time r_{xy}

or less than the current time t, then F_{xy} cannot meet its deadline, and thus, is discarded (and a dynamic failure occurs).

Note that, when a video stream S_i is requested to be transmitted (assuming its first frame F_{i1} is ready at time r_{i1}), then at time r_{i1} we will (i) pre-schedule the urgent frames (in this case, only I-frames) of the next H_i frames (including F_{i1}) in S_i, (ii) reschedule those pre-scheduled urgent frames that will be affected by these newly pre-scheduled urgent frames, and (iii) record/update their notification times.

In summary, we will always look ahead H_i frames (including the current frame) when pre-scheduling the urgent frames of video stream S_i. The lookahead number H_i ($1 \leq H_i \leq Q_i$) is a user-specified and/or application-dependent parameter. It is easy to see that a larger H_i will result in a better performance (i.e., lower probability of dynamic failure and fewer frame deadline misses), but the scheduling overhead will be larger.

If a frame F_{ij} completes its transmission before its deadline, and the past $K_i - 2$ frames, $F_{i,j-1}, F_{i,j-2}, \ldots, F_{i,j-K_i+2}$, have all met their deadlines, then stream S_i leaves the urgent state and moves back to the normal state.

During the transmission of video streams, if the notification time v_{ij} of an urgent frame F_{ij} is reached, the server activates the transmission of F_{ij} by preempting any normal or urgent frame currently being transmitted. That is, every urgent frame, if activated on or after its notification time, has a higher priority than all normal frames and other urgent frames being transmitted before their notification times. The priorities of urgent frames that are transmitted on or after their notification times are defined by the backwards-EDF algorithm. If F_{ij} preempts another urgent frame F_{xy} currently being transmitted before its notification time, then we must adjust the notification times of F_{xy} as well as some other pre-scheduled urgent frames (details of this case will be given in Section 3.2). If F_{ij} preempts another urgent frame F_{xy} currently being transmitted on or after its notification time, then we push the id, (x, y), of F_{xy} onto a stack, called the *preemption stack*. Later when the transmission of F_{ij} (or some other urgent frame) is completed and we want to resume the transmission of a preempted urgent frame, we will pop up the frame id from the top of the preemption stack.

For example, suppose urgent frames F_1-F_4 are pre-scheduled using the backwards-EDF algorithm as shown in Fig. 6, and F_4 begins its transmission at time t_1. At time t_2, F_4 will be preempted by F_2 and F_4's frame id, 4, will be pushed onto the preemption stack. F_2 is then transmitted from time t_2 to time t_3, and preempted by F_1 at time t_3. F_2's frame id, 2, is pushed onto the preemption stack at time t_3, and F_1 is transmitted from time t_3 to time t_4. When F_1 completes its transmission at time t_4, we pop an id from the preemption stack, which will be 2, and resume the transmission of F_2. When F_2 completes its transmission at time t_5, we will begin the transmission of F_3 since its notification time $v_3 = t_5$ is reached. When F_3 completes its transmission at time t_6, we pop an id from the preemption stack, which will be 4, and resume the transmission of F_4. F_4 will complete its transmission at time t_7.

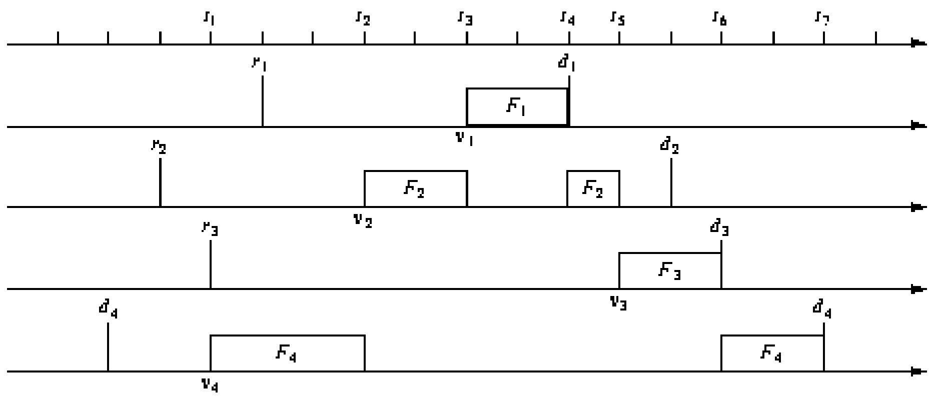

Figure 6: The transmission of pre-scheduled urgent frames.

By using the preemption stack to store the id's of the urgent frames transmitted after their notification times and preempted by other urgent frames, and later by popping up their id's from the stack to resume their transmission, we actually transmit the pre-scheduled urgent frames according to the backwards-EDF algorithm. This is proved in the following theorem and corollary.

Theorem 1: The allocation intervals in a schedule produced by any priority-driven preemptive scheduling algorithm is properly nested, i.e., if F_{ij} has a higher priority than F_{xy}, then either $s_{xy} < s_{ij} < f_{ij} < f_{xy}$ or the time intervals $[s_{xy}, f_{xy})$ and $[s_{ij}, f_{ij})$ do not overlap, where s_{ij} (s_{xy}) and f_{ij} (f_{xy}) are the start time and finish time, respectively, of the transmission of F_{ij} (F_{xy}) in the schedule.

Proof: The transmission of a higher-priority frame F_{ij} started at time s_{ij} will not be preempted by any lower-priority frame F_{xy} before the completion of its transmission at time f_{ij}, and thus, we have $s_{ij} < f_{ij} \leq s_{xy} < f_{xy}$. If F_{xy} is not preempted by a higher-priority frame F_{ij} during its transmission, the start time s_{ij} of F_{ij}'s transmission is no less than the finish time f_{xy} of F_{xy}. Therefore, $s_{xy} < f_{xy} \leq s_{ij} < f_{ij}$. On the other hand, if F_{xy} is preempted by a higher-priority frame F_{ij} before the completion of its transmission, then before F_{ij} finishes its transmission, F_{xy} will not resume its transmission. Therefore, $s_{xy} < s_{ij} < f_{ij} < f_{xy}$.

Note that the above argument is true regardless of whether or not there are frames other than F_{ij} and F_{xy} being transmitted during time intervals $[s_{ij}, f_{ij})$ and $[s_{xy}, f_{xy})$, respectively. □

Since backwards-EDF is a priority-driven preemptive scheduling algorithm, the following corollary follows directly from the above theorem.

Corollary 1: By using the preemption stack as described above, the pre-scheduled urgent frames are transmitted according to the schedule produced by the backwards-EDF algorithm. □

During the transmission of the video streams, if, at a certain time t, no normal frame is waiting for transmission and no urgent frame notification time has been reached, instead of leaving the server idle, one of the pre-scheduled urgent frames, say F_{ij}, will be chosen for transmission. If a normal frame becomes ready or the notification time of another urgent frame is reached during the transmission of an urgent frame F_{ij} prior to its notification time, F_{ij} is preempted and its notification time needs to be adjusted to reflect the remaining frame size. Moreover, the notification times of some other pre-scheduled urgent frames may also need to be adjusted accordingly. However, if the frame F_{ij} chosen to be transmitted prior to its notification time is the one with the earliest notification time among all the active urgent frames at time t, we will show (Theorem 2 in Section 3.2) that no other urgent frame notification time needs to be adjusted when F_{ij}'s notification time is adjusted (since the notification time of any other pre-scheduled urgent frame won't be affected by the change of the remaining frame size of F_{ij}). We call this service policy the *earliest-notification-time-first* (ENF) policy.

As a result, each pre-scheduled urgent frame has two transmission modes: *pre-notification* and *post-notification*. An urgent frame transmitted before its notification time is said to be in its pre-notification mode and receives a lower priority than all normal frames. An urgent frame transmitted on and after its notification time is said to be in its post-notification mode and has a higher priority than any normal frame (and hence, has a higher priority than any urgent frame in its pre-notification mode).

We summarize below the transmission of a normal frame. (The transmission of urgent frames in their pre- and post-notification modes are detailed in Section 3.2.)

When a normal frame F_{ij} is being transmitted, several events may occur. Suppose at time s, F_{ij} begins its transmission and at time $t > s$, then one of the following events occurs (and between time s and t, none of the events occurs).

NORM1: The notification time of a pre-scheduled urgent frame F_{xy} is reached, i.e., $t = v_{xy}(t)$.

F_{xy} preempts F_{ij}, and the server switches to transmit F_{xy} in its post-notification mode. Since the remaining frame size of F_{ij} has decreased by $t - s$, the remaining frame size of F_{ij} is updated as $C_{ij}(t) = C_{ij}(s) - (t - s)$.

NORM2: Another normal frame F_{xy} becomes ready for transmission, i.e., $t = r_{xy}$.

If F_{ij} has a higher priority than F_{xy} (i.e., $d_{ij} \leq d_{xy}$), we continue the transmission of F_{ij}; otherwise, F_{ij} will be preempted by F_{xy}, and the server switches to transmit F_{xy}. But, before starting the transmission of F_{xy}, we first check if $C_{xy}(t) \leq d_{xy} - t - R(t, d_{xy})$, where $R(t, d_{xy})$ is the amount of time that has been reserved for urgent frames from time t to time d_{xy} (at time t). If yes, we transmit F_{xy}, and the remaining frame size of F_{ij} is updated as $C_{ij}(t) = C_{ij}(s) - (t - s)$. Otherwise, we discard F_{xy}, pre-schedule the urgent frames (the next $H_x - 1$ frames in S_x) and reschedule some other urgent frames, record/update their notification times, and then, return to transmit F_{ij}.

NORM3: F_{ij} finishes its transmission.

If there are other normal frames waiting for transmission (i.e., there are active normal frames at time t), choose the one, say F_{xy}, with the earliest deadline to transmit next. Before starting the transmission of F_{xy}, we first check if $C_{xy}(t) \leq d_{xy} - t - R(t, d_{xy})$, where $R(t, d_{xy})$ is the amount of time that has been reserved at time t for urgent frames from time t to time d_{xy}. If yes, we transmit F_{xy}. Otherwise, we abort/discard F_{xy}, pre-schedule the urgent frames of the next $H_x - 1$ frames in S_x and reschedule some other urgent frames, record/update their notification times, and then, repeat this process (**NORM3**).

If there is no other normal frame waiting for transmission, but there are active urgent frames, we choose among all the active urgent frames the one with the earliest notification time to transmit in its pre-notification mode.

Finally, if there is neither active normal frame nor active urgent frame at time t, we just leave the server idle.

The Proposed Transmission Scheduling Scheme

Pre- and post-notification transmission modes

When an urgent frame F_{ij} is being transmitted in its pre- or post-notification mode, the remaining frame size of F_{ij} decreases with the progress in its transmission, and when F_{ij} is preempted by some other frame it will be rescheduled with the remaining frame size, and the notification time v_{ij} of F_{ij} changes accordingly. Therefore, in the following discussion, we use $v_{ij}(t)$ to denote the value of v_{ij} at time t.

When there is no normal frame waiting for transmission and none of the notification times of the pre-scheduled urgent frames has been reached, one of the pre-scheduled active urgent frames will be chosen for transmission in its pre-notification mode. Recall that a frame F_{ij} is said to be active at time t if $r_{ij} \leq t < d_{ij}$ and $C_{ij}(t) > 0$, and for each video stream S_i there is at most one active frame at any time t. Therefore, there are at most n active frames at any given time.

In choosing pre-scheduled urgent frames to transmit in their pre-notification mode, it is better to use the ENF policy, i.e., always choose the active urgent frame with the earliest notification time to transmit first. Therefore, if at time s, we need to choose an active urgent frame to transmit in its pre-notification mode, we will choose the one, say F_{ij}, with the earliest notification time $v_{ij}(s)$ among all active urgent frames at time s . However, F_{ij}'s notification time will increase with the progress of its transmission, and at a certain time $t > s$, its notification time may become larger than that of some other active urgent frame, say F_{xy}. Since we want to use the ENF policy in choosing an urgent frame to transmit in its pre-notification mode, we should switch to transmit F_{xy} when the notification time of F_{ij} becomes larger than that of F_{xy}. Hence, we must be able to detect when such a situation happens. The following definition is introduced for this purpose.

Let $v_{pq}(s)$ be the smallest notification time among all urgent frames, except F_{ij}, at time s such that $v_{ij}(s) < v_{pq}(s) < d_{ij}$. If such a notification time exists (from Theorem 1, we know that F_{pq} has a higher priority than F_{ij}), define $v = v_{pq}(s)$; otherwise, define $v = d_{ij}$. We then transmit frame F_{ij} in its pre-notification mode for at most $(v - v_{ij}(s))$ units of time. When F_{ij} is being transmitted in its pre-notification mode, several events may occur. Suppose at time $t > s$, one of the following four events occurs (and between time s and t, none of the events occurs).

PRE1: The notification time of another pre-scheduled urgent frame F_{xy} is reached, i.e., $t = v_{xy}(t)$.

> F_{ij} will be preempted by F_{xy}, and the server switches to transmit F_{xy} in its post-notification mode. Since the remaining frame size of F_{ij} has decreased by $t - s$ (i.e., $C_{ij}(t) = C_{ij}(s) - (t - s)$), its notification time should be increased by $t - s$. Therefore, its notification time is updated as $v_{ij}(t) = v_{ij}(s) + t - s$. It will be shown (in Theorem 2) that only the notification time of F_{ij} needs to be adjusted.

PRE2: A normal frame F_{xy} becomes ready (active), i.e., the ready time of F_{xy} is reached ($t = r_{xy}$).

> F_{ij} will be preempted by F_{xy}, and its remaining frame size and notification time are updated/adjusted as in **PRE1**. But, before starting the transmission of F_{xy}, we first check if $C_{xy}(t) \leq d_{xy} - t - R(t, d_{xy})$, where $R(t, d_{xy})$ is the amount of time that has been reserved at time t for urgent frames from time t to time d_{xy} (after the notification time v_{ij} of F_{ij} is adjusted). If yes, we transmit F_{xy}. Otherwise, we discard F_{xy}, pre-schedule the urgent frames (the next $H_x - 1$ frames) in S_x and reschedule some other urgent frames, record/update their notification times, and then, choose among all active urgent frames the one with the earliest notification time to transmit next.

PRE3: A pre-scheduled urgent frame F_{xy} becomes ready (active).

> Update F_{ij}'s remaining frame size and notification time as in **PRE1**, and then, choose between F_{ij} and F_{xy} the one with the earlier notification time to transmit next.

PRE4: F_{ij} has been transmitted for $v - v_{ij}(s)$ units of time.

> The equality $v = d_{ij}$ means that F_{ij} completes its transmission before its deadline and the notification time of F_{ij} can now be cancelled since it is no longer needed. Moreover, if the past $K_i - 2$ frames, $F_{i,j-1}, F_{i,j-2}, \ldots, F_{i,j-K_i+2}$, have all met their deadlines, then stream S_i leaves the urgent state and moves back to the normal state. Then, we choose at time t the frame with the earliest notification time among all active urgent frames to transmit next.
>
> On the other hand, if $v = v_{pq}(s)$, we reschedule F_{ij} (with remaining frame size $C_{ij}(t) = C_{ij}(s) - (t - s) = C_{ij}(s) - (v - v_{ij}(s))$) and adjust its notification time accordingly. Then, again, we choose the frame with the earliest notification time among all active urgent frames at time t to transmit next.

Theorem 2: In events **PRE1-PRE4**, F_{ij} is the only frame whose notification time has changed and needs to be recalculated and adjusted at time t.

Proof: It is easy to see that only the notification times of the frames with priorities lower than that of F_{ij} will be affected by the reduced remaining frame size of F_{ij}. According to the backwards-EDF algorithm, a frame

F_{xy} with a priority lower than F_{ij} has a ready time $r_{xy} \leq r_{ij}$. Since F_{ij} is active (ready) at time s, $r_{ij} \leq s$. Therefore, F_{xy} is also ready at time s ($r_{xy} \leq s$). If we choose F_{ij}, instead of F_{xy}, to transmit at time s, it implies that $v_{ij}(s) < v_{xy}(s)$. From Theorem 1, we know that $d_{ij} \leq v_{xy}(s) = v_{xy}(t)$, meaning that the change in the remaining frame size and the notification time of F_{ij} will not affect the notification time of F_{xy}. □

When a frame F_{ij} is transmitted in its post-notification mode starting from time s, there are two events that may occur. Suppose at time $t > s$, one of the following two events occurs (and between time s and t, neither of the two events occurs).

POST1: Another higher-priority frame F_{xy}'s notification time v_{xy} is reached, i.e., $t = v_{xy}(t)$.

> F_{ij} should be preempted and the server should switch to transmit F_{xy}. F_{ij}'s frame id, i.e., (i, j), is pushed onto the preemption stack.

POST2: F_{ij} completes its execution.

> The notification time of F_{ij} can now be cancelled since it is no longer needed. Also, if the past $K_i - 2$ frames, $F_{i,j-1}, F_{i,j-2}, \ldots, F_{i,j-K_i+2}$, have all met their deadlines, then stream S_i leaves the urgent state and moves back to the normal state. Then, depending on the situation, the server should: (1) transmit an urgent frame F_{xy} in its post-notification mode if its notification time $v_{xy}(t)$ equals t; (2) pop up the top element (x, y) from the preemption stack if the stack is nonempty, and resume the transmission of frame F_{xy} in its post-notification mode; (3) transmit a normal frame (according to the EDF policy) if the preemption stack is empty and some normal frames are waiting for transmission; (4) transmit the active urgent frame (in its pre-notification mode) with the earliest notification time among all active urgent frames at time t if the preemption stack is empty and no normal frame is waiting for transmission; (5) be left idle if there are no active (urgent/normal) frames at time t.

The Proposed Transmission Scheduling Scheme

Table of Contents

Justifications of the Proposed Scheme

The EDF service policy is optimal in the sense that if a set of frames is schedulable by any other service policy, then it is also schedulable by EDF. In our transmission scheduling scheme, normal frames are transmitted according to the EDF policy. Thus, if a normal frame F_{ij} cannot meet its deadline d_{ij}, then using any other service/scheduling policy for the same set of video streams, there must be a frame F_{xy} which also cannot meet its deadline d_{xy}. Moreover, it is easy to see that $d_{xy} \leq d_{ij}$. Therefore, using any other service policy for normal frames, the system will enter the urgent state no later than the case of using the EDF policy. This justifies our choice of EDF as the policy for transmitting normal frames.

When a frame F_{ij} is found unable to meet its deadline, our scheme simply discards/aborts it. The next $K_i - 1$ frames of S_i then become urgent because missing the deadline of any one of these frames will result in a dynamic failure. A good transmission scheduling scheme must therefore raise the priorities of these urgent frames. In the approach proposed in [3], they simply raise the priorities of the urgent frames to a level higher than all normal frames, and then transmit urgent frames before transmitting normal frames. By contrast, in our scheme we ensure the timely transmission of urgent frames by pre-scheduling them. Since we pre-allocate the server time to urgent frames to ensure their timely transmission, there is no need to schedule them too early. This is why we use the backwards-EDF algorithm to pre-schedule the urgent frames as late as possible.

It can be shown that using backwards-EDF, we will leave the largest room for transmitting normal frames in the front part of the schedule. Let $\Omega_{\mathbf{S}}^{X}(t_1, t_2)$ denote the total amount of idle time in the interval [t_1, t_2] of the schedule for the set of frames $\mathbf{S}$ produced by a scheduling algorithm X. Chetto *et al.* [1] proved that the schedule produced by the EDF algorithm satisfies the inequality $\Omega_{\mathbf{S}}^{\mathrm{EDF}}(0,t) \le \Omega_{\mathbf{S}}^{X}(0,t)$ for any preemptive scheduling algorithm X and at any time t. In fact, it can easily be shown that $\Omega_{\mathbf{S}}^{A}(0,t) \le \Omega_{\mathbf{S}}^{X}(0,t)$ is true for any *work-conserving* [11] transmission scheduling algorithm A and any preemptive scheduling algorithm X, since in any work-conserving algorithm, the server is never left idle when there are frames ready for transmission. Since any priority-driven scheduling algorithm is work-conserving, it implies that $\Omega_{\mathbf{S}}^{B}(0,t) \ge \Omega_{\mathbf{S}}^{X}(0,t)$, where B denotes the backwards-A algorithm for any priority-driven scheduling algorithm A (X denotes any preemptive scheduling algorithm). That is, the backwards-EDF algorithm leaves as much idle time as possible for the possible transmission of normal frames before the transmission of any pre-scheduled urgent frames. This justifies our choice of backwards-EDF as the pre-scheduling algorithm for urgent frames.

In our scheme, if there is no normal frame waiting for transmission, we will always choose the active urgent frames with the earliest notification time to transmit, and hence still leave as much room as possible for future normal frames. This justifies the use of pre-notification transmission mode for urgent frames.

It is easy to see that our scheme has good performance in terms of reducing the probability of dynamic failure and meeting more frame deadlines because it utilizes prior knowledge of future frame sizes and deadlines. However, the amount of this prior knowledge may depend on the underlying multimedia/communication application. Note that the server knows at least the size and deadline of the frame currently waiting for transmission, i.e., $Q_i = 1$ and d_{ij} is known when F_{ij} is ready. Therefore, our scheme is applicable to all applications by setting the lookahead number H_i equal to 1 for all i. Even with $H_i = 1\ \forall i$, our scheme performs better than other schemes that do not utilize any knowledge of frame sizes and deadlines, such as the service policy proposed in [3].

Table of Contents

Conclusion

In this paper, we formulated the problem of scheduling the transmission of MPEG-compressed video streams with firm deadline constraints on a single server, and proposed an effective scheduling scheme which can transmit as many frames as possible before their deadlines while reducing the probability of dynamic failure, where a dynamic failure is either an I-frame deadline miss or two or more frame deadline misses within a certain number of consecutive frames in a video stream. The proposed scheme uses the last-chance philosophy and a pre-scheduling approach to pre-allocate the server time to a set of urgent frames as late as possible so that non-urgent frames will not be blocked by urgent frames, and thus, have a better chance to meet their deadlines.

Our scheme uses the lookahead number H_i to specify how many urgent frames will be pre-scheduled. As mentioned in Section 3.1, it is easy to see that a larger H_i will result in better performance, but the scheduling overhead will also be larger. We are currently working on the analysis/simulation of the effects of changing the lookahead number. Further results on this will be reported in a forthcoming paper.

Finally, we would like to emphasize that although our problem formulation and transmission scheduling scheme are motivated by MPEG-coded video transmission, they are applicable to the transmission of general (not necessarily MPEG-coded) video/message streams.

Table of Contents

End Notes

The work reported in this paper was supported in part by the ONR under Grants N00014-92-J-1080 and N00014-94-1-0229, and by the NSF under Grant MIP-9203895.

[Table of Contents](#)

Acknowledgment

The authors would like to thank Jennifer Rexford of the Real-Time Computing Laboratory for her critical comments on an early draft of this paper.

[Table of Contents](#)

References

1

Houssine Chetto and Maryline Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Trans. on Software Engineering*, 15(10):1261-1269, October 1989.

2

Didier Le Gall. Mpeg: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46-58, April 1991.

3

M. Hamdaoui and P. Ramanathan. A service policy for real-time customers with (m, k)-firm deadlines. In *Proc. of the 24th Annual Int'l Symposium on Fault-Tolerant Computing*, pages 196-205, 1994.

4

Ching-Chih Han and Kang G. Shin. A globally optimal algorithm for scheduling both hard periodic and soft aperiodic tasks. submitted to *IEEE Trans. on Computers*, 1994.

5

Simon S. Lam, Simon Chow, and David K.Y. Yau. An algorithm for lossless smoothing of MPEG video. In *Proc. of ACM SIGCOMM 94*, pages 281-293, London England UK, 1994.

6

C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. of ACM*, 20(1):46-61, Jan. 1973.

7

Coding of moving pictures and associated audio. SC29/WG11 Committee (MPEG) Draft of Standard ISO-IEC/JTC1 SC29, November 1991.

8

Teunis Ott, T. V. Lakshman, and Ali Tabatabai. A scheme for smoothing delay-sensitive traffic offered to ATM networks. In *Proc. of IEEE INFOCOM'92*, pages 776-785, 1992.

9

Pramond Pancha and Magda El Zarki. A look at the MPEG video coding standard for variable bit rate video transmission. In *Proc. IEEE INFOCOM'92*, pages 85-94, 1992.

10

Pramond Pancha and Magda El Zarki. Bandwidth requirements of variable bit rate MPEG sources in ATM networks. In *Proc. IEEE INFOCOM'93*, pages 902-909, 1993.

11

Craig Partridge. *Gigabit Networking*. Reading, Mass.: Addison-Wesley, 1994.

12

A. R. Reibman and A. W. Berger. On VBR video teleconferencing over ATM networks. In *Proc. of IEEE GLOBECOM'92*, pages 314-319, 1992.

13

D. Reininger, D. Raychaudhuri, B. Melamed, B. Sengupta, and J. Hill. Statistical multiplexing of VBR MPEG compressed video on ATM networks. In *Proc. of IEEE INFOCOM'93*, 1993.

Table of Contents