Rate-Controlled Scheduling for Networked Multimedia Applications *

Fan Zhang and Samuel T. Chanson

Department of Computer Science

Hong Kong University of Science and Technology

Clear Water Bay, Kowloon, Hong Kong

Email: {zhangfan, chanson}@cs.ust.hk

Abstract

Multimedia applications over the Internet are becoming increasingly popular. The soft real-time constraints of this type of applications have given rise to interesting issues for scheduling in the network as well as the end systems. This paper presents a rate-based task scheduling framework for the receiving hosts. As multimedia streams are expected to be played back at constant rate, the proposed scheme tries to maintain the specified output frame rate by assigning proper deadlines to the frames. A multimedia stream that falls behind can catch up by skipping some frames. Interference among tasks is prevented using the rate-controlled servers (RCS). The performance of these schemes have been analyzed mathematically and evaluated by simulations.

1 Introduction

With the increased capability of today's computers and network connectivity, multimedia applications over the Internet are becoming increasingly popular. New applications such as Internet telephony, videoconferencing and video-on-demand have made their way into homes and offices. These types of multimedia applications involve transmission and processing of continuous media (CM) streams such as audio and video. CM streams are sensitive to loss, delay and delay jitter, and therefore should be delivered, processed and played back in a timely manner to meet end-to-end quality of service (QoS) requirements.

Each media frame must be generated, transmitted and then processed at the receiving host for playback (see Figure 1). Many algorithms for packet scheduling in the network nodes have been proposed to regulate network delay and smoothen delay jitters (see [1] for example). This paper, however, studies processor scheduling for multimedia applications at the end host. Although processor scheduling at the end host cannot fully solve the problem created in the network, if reasonable network QoS can be maintained, appropriate processor scheduling algorithms will help to alleviate network distortion and

Figure 1: End-to-End process of videoconferencing.

produce smoother playback.

CM streams need to be played back at a constant rate. It is a natural approach to deploy a rate-controlled scheduler to process the arriving frames. In the context of hard real-time systems, Liu and Layland's periodic task model [2] has been extensively studied, however, new issues arise when networked multimedia applications are considered:

- Although multimedia frames are released at a fixed rate, the arrival pattern at the destination is often distorted due to network congestion and traffic variation in the network. Task models assuming particular arrival patterns such as the periodic task model [2] is therefore not applicable.
- The application at the destination host should decide the playback time and assign processing deadline to each frame. Due to the distorted frame arrival pattern, simply applying a deadline relative to the frame arrival times will produce an irregular deadline pattern.
- Multimedia applications can tolerate certain quality degradation like frame loss, which can be exploited to reduce end-to-end delay.
- Multimedia applications are often served by a general-purpose system on which other types of applications (hard real-time and non-real-time) also run. Task isolation is required to prevent misbehaving applications from interfering with the conforming ones.
- Providing a buffer at the end host to regulate the playback rate may not be suitable for interactive applications which demand very short end-to-end delay.

In this paper, we present a new task model called rate-controlled processing (RCP) for multimedia streams over the Internet. This model does not make any assumption on the minimum inter-arrival time of the processing requests. However, the average inter-arrival time is assumed to be equal to the frame period, which is the time between successive frame generation. Deadlines are computed based on the specified frame rate and the frame arrival patterns at the destination host, and are assigned to the frames on arrival. In this way, our approach tries to compensate for the arrival jitters introduced during network

---

*The work described in this paper was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. HKUST6077/01E).

transmission, thereby helping to minimize the end-to-end delay and enforce the expected processing rate. Schedulability conditions are derived to ensure the assigned deadlines will not be violated. For multimedia applications that do not require hard guarantee, RCP has a frame skip option to trade frame loss for shorter delay. In an open environment where different types of applications co-exist, we propose the rate-controlled server (RCS) to achieve task isolation and provide guaranteed service for hard real-time tasks.

The rest of this paper is organized as follows: In section 2, we review some related CPU scheduling algorithms for multimedia applications. In section 3, we present the RCP task models and derive the schedulability conditions. The RCS model is also introduced in this section based on the RCP framework. Simulation results are presented and discussed in section 4. Finally, section 5 concludes the paper.

2 Related Work

Recently, some rate-controlled scheduling approaches have been proposed, which offer real-time guarantee and provide certain degree of fairness. Among these are proportional sharing and reservation-based sharing.

The proportional sharing frameworks [3, 4] are similar to the weighted fair queuing (WFQ) for network packet scheduling. A weight is associated with each stream to denote its execution requirements. The scheduling algorithm allocates processing times proportional to the weights of the streams. In the earliest eligible virtual deadline first (EEVDF) algorithm [3], a quantum is assigned to the stream with the earliest virtual deadline¹. The start-time fair queuing (SFQ) scheme [4] uses start tags instead of virtual deadlines, thereby removing the need to know the processing time requirement of each frame.

In reservation-based sharing [5, 6], each stream specifies its execution period and reserves the required bandwidth. Budgets are allocated to the streams according to their reservations and deadlines are assigned. As a stream is processed, its budget is continuously consumed. The scheduler selects the stream with the earliest deadline and a non-zero budget for execution. The constant utilization server (CUS) [5] and the constant bandwidth server (CBS) [6] schemes belong to this category and differ only in the way deadlines are calculated and budgets are replenished. In a recent work [7], Lipari and Baruah extended CBS to a two-level hierarchy, which allows bandwidth sharing among the threads in the same application.

Ideally frames should be completed and played back at constant rate in multimedia applications. However, both the proportional sharing and reservation-based approaches only guarantee specific amount of CPU time but do not consider the output frame rate which directly affects the perceived quality of multimedia streams. As the processing times change from period to period and the frame arrival rates are irregular, these approaches cannot guarantee the expected output frame rate for networked multi-

---

¹The virtual deadline of a stream at time t is the time point by which the same amount of accumulated CPU time allocated to stream at t would have been allocated in an ideal fluid flow system.

media applications.

Jeffay et al. proposed the rate-based execution (RBE) model [8]. An RBE process [8] is described by the number of frames to be processed in an interval and a response time (also called relative deadline). The response time is independent of the execution rate and specifies the desired maximum elapsed time between the arrival of a frame and the completion of processing that frame. Deadlines are assigned in accordance with the specified processing rate. Although RBE introduced the concept of job processing rate, it forces a *constant* distance (called response time) between job arrival and its deadline and thus cannot dynamically alleviate delay and delay jitters.

3 Task Models

A *task* in our work is characterized by a sequence of CPU requests. Each request is called a *job*. A job is said to *meet* its deadline if the job is completed before a given deadline. Otherwise we say that the job *misses* its deadline. Consider a video stream. In our model, the stream is a task and each video frame is a job. Formally, we define a task T_i as a sequence of jobs $J_{i,1}, J_{i,2}, J_{i,3} \dots J_{i,n}$. Each job is described by a 3-tuple:

$$J_{i,j} = (r_{i,j}, e_{i,j}, d_{i,j}),$$

where $r_{i,j}$ denotes the release time; $e_{i,j}$ denotes the execution time (the shadowed part) and $d_{i,j}$ denotes the deadline of job $J_{i,j}$. For a job processing a video or audio frame, its release time is the arrival time of the frame, and the execution time is the required processing time of the frame. These times are known when the frame arrives at the destination host and cannot be changed. However, the deadline is flexible and needs to be assigned by the system.

In some existing systems, processed frames are stored in a buffer, and then taken out and played back sequentially. The frame processing deadline and its playback time are not directly correlated. Therefore it is hard to tell the playback quality when frames are scheduled to be processed.

We propose a new approach which couples the frame processing deadline and the frame playback time. When a frame arrives at the end host, a *frame deadline* is assigned by the application, which is the frame playback time. As a frame must be completely processed before it can be played back, it is natural to use this frame deadline also as the deadline of the job processing the frame. Here we assume the delay of playing back a decoded frame is negligible. From the user's perspective, the smoothness of the stream is determined by the smoothness of the frame playback sequence, therefore the frame deadline pattern in our model. As every frame will be played back exactly at its frame deadline, we can guarantee the specified QoS by assigning proper frame deadlines and ensuring that jobs meet the assigned deadlines. In the following subsections, we shall introduce the RCP model and derive the feasibility conditions for implementing admission control.

3.1 Rate-controlled Processing

3.1.1 Deadline Assignment

As mentioned previously, multimedia frames are expected to be processed and played back at a constant rate. RCP implicitly controls the frame rate by regulating the frame deadlines and smoothes out the delay jitter introduced in the network.

Formally, we characterize a RCP task with a 3-tuple (L, E, p) . L is called the *lead time* of the task, that is, the interval between the deadline and the arrival time of the first job in the task. E is the *worst-case execution time* (WCET) of any job in the task. p is the *frame period* which is the reciprocal of the frame rate. The meaning of the frame period is manifold: first, at the source host, released frames are spaced by the frame period; second, it equals the long-term average frame arrival period at the destination host (although the average interval in a short period of time may differ from p); third, it denotes the desired processing and playback rates, i.e., one frame should be processed and played back in each frame period.

Suppose $r_{i,j}$ is the arrival time of the j th frame of stream i , then $r_{i,j}$ is also the release time of the job processing the frame $(J_{i,j})$. The deadline of $J_{i,j}$ is assigned according to the following formula:

$$d_{i,j} = \begin{cases} r_{i,j} + L_i & \text{if } j = 1 \\ \max(r_{i,j} + \min(L_i, p_i), d_{i,j-1} + p_i) & \text{if } j > 1 \end{cases} \quad (1)$$

The deadline of the first job of each task is calculated by adding the lead time to the arrival time. The deadline of every subsequent job takes both the deadline of the previous job and the specified execution rate into consideration. Although consecutive jobs may arrive in a burst, their deadlines are spaced out by the period p . This deadline assignment implicitly controls the playback rate and enforces the actual processing rate at the specified frame rate.

The RCP model can be regarded as a generalization of the traditional periodic task model [2]. If the jobs of a RCP task are released at a constant rate of one per period, then the deadlines assigned are exactly the same as those of a periodic task with the same release time, relative deadline (i.e., the lead time in our model) and period.

The value of the lead time L_i can be larger or smaller than the task period p . The decision is made based on two considerations: the end-to-end delay requirement and the network delay jitter condition. A shorter lead time tends to reduce the end-to-end delay while a longer one will help smooth out the network delay jitter. For interactive multimedia applications, the lead time can be decided at the session set up time by sending out probing packets and estimating the network delay and delay jitter. Generally speaking, the lead time should be chosen so that the total end-to-end delay of the frame at playback will not exceed the specified limit. The limit is application dependent and also somewhat depends on what the end user is willing to tolerate. Within this constraint, the longer the lead time (often larger than the frame period), the smoother the playback quality.

3.1.2 Feasibility of Preemptive Scheduling

In this work, we assume the tasks are fully preemptible. An RCP task set is feasible if and only if all the jobs in the task set can meet their deadlines. Due to space limitation, some of the proofs are omitted. The reader is referred to [9] for details.

Theorem 1 (Dertouzos) *The EDF algorithm is optimal in that if there exists any algorithm that can build a valid (feasible) schedule on a single processor, then the EDF algorithm can also build a valid schedule.*

Dertouzos [10] has shown that preemptive EDF is optimal for jobs with arbitrary release times, deadlines and execution times. This means a task set is schedulable if and only if it is schedulable with EDF. Without loss of generality, we only need to analyze the feasibility of a task set with respect to EDF.

Definition 1 *The processor demand of a task set during interval $[t_1, t_2]$ is given by*

$$pd_{[t_1, t_2]} = \sum_{t_1 \leq r_{i,j}, d_{i,j} \leq t_2} c_{i,j},$$

i.e., the sum of the processing times required by all the jobs in the set of tasks with arrival times at or after t_1 and deadlines at or before t_2 .

Definition 2 *The interval utilization of a task set during an interval $[t_1, t_2]$ is*

$$intU_{[t_1, t_2]} = \frac{pd_{[t_1, t_2]}}{t_2 - t_1}.$$

Theorem 2 ([11]) *A real-time task set can be feasibly scheduled by EDF if and only if the following inequality holds:*

$$\sup_{0 \leq t_1 < t_2} intU_{[t_1, t_2]} \leq 1.$$

Since t_1 and t_2 can assume any value satisfying the condition $0 \leq t_1 < t_2$, it is not possible to calculate the interval utilization of each interval to determine $\sup_{0 \leq t_1 < t_2} intU_{[t_1, t_2]}$. Fortunately, the processor demand of a given RCP task set in any interval of a given length can be bounded, as indicated in the following Lemma.

Lemma 1 *Given an RCP task with specification (L_i, E_i, p_i) , during any interval of length $Len > 0$, the least upper bound on the processor demand during the interval is $\left\lfloor \frac{Len + p_i - \min(p_i, L_i)}{p_i} \right\rfloor \cdot E_i$.*

Theorem 3 *A given RCP task set is schedulable by EDF if and only if for $\forall Len > 0$,*

$$pd_{Len} = \sum_{i=1}^n \left\lfloor \frac{Len + p_i - \min(p_i, L_i)}{p_i} \right\rfloor \cdot E_i \leq Len.$$

Theorem 3 gives the sufficient and necessary conditions for an RCP task set to be schedulable with EDF. Note that by Theorem 1, it can be shown that the condition given in Theorem 3 is also sufficient and necessary for any algorithm to feasibly schedule a rate-controlled task set. The proofs of Lemma 1 and Theorem 3 can be found in [9].

The above results are based on the processor demand approach and provide a method to evaluate the feasibility of a given RCP task set. Using the technique given by Baruah et al. [12], it is sufficient to test the processor demand for a finite number of intervals if the total processor utilization $\sum_{i=1}^n (\frac{E_i}{p_i})$ is strictly less than 1. Thus the feasibility test has pseudo-polynomial complexity. On the other hand, if $\forall i, L_i \geq p_i$, then the task set is schedulable if and only if $\sum_{i=1}^n (\frac{E_i}{p_i}) \leq 1$, which is the same as the utilization bound of Liu and Layland's periodic task model [2]. Finally we give another simple sufficient condition for RCP tasks. Unfortunately, this condition is not necessary.

Theorem 4 *Given an RCP task set, if*

$$\sum_{i=1}^n \frac{E_i}{\min(L_i, p_i)} \leq 1,$$

then the task set is schedulable.

Proof: We use the processor demand approach. The least upper bound of processor demand during an interval of length Len is

$$\begin{aligned} pd_{Len} &= \sum_{i=1}^n \left\lfloor \frac{Len + p_i - \min(p_i, L_i)}{p_i} \right\rfloor \cdot E_i \\ &\leq \sum_{i=1}^n \left(\frac{Len}{\min(L_i, p_i)} \right) \cdot E_i \\ &= \sum_{i=1}^n \left(\frac{E_i}{\min(L_i, p_i)} \right) \cdot Len \\ &\leq Len. \end{aligned}$$

By Theorem 3, the task set is schedulable. □

3.1.3 RCP with Skips

If the schedulability condition is satisfied, all frames will be processed before their deadlines in the basic RCP model. However, when a frame arrives late, the frame deadline will be postponed, giving it enough time to be processed. However, this will increase the end-to-end delay of the frame. Furthermore, as consecutive frame deadlines must be spaced by at least a period, the end-to-end delay will never decrease. For interactive multimedia applications like videoconferencing with stringent end-to-end delay constraint, this approach is obviously not satisfactory.

Fortunately, most multimedia applications can tolerate occasional frame loss. We say a stream is *K-skipppable* if it can tolerate 1 frame loss out of K consecutive frames. For applications that have stringent

delay requirements, we can skip² some frames to reduce the end-to-end delay of subsequent frames. We use a 2-tuple (A, P) to specify the end-to-end delay requirements. ‘ A ’ denotes *Acceptable delay* which is the longest tolerable delay; while ‘ P ’ denotes *Preferred delay*, which represents a higher quality level. We assume the release time of a frame at the source host is known by checking the timestamp of the frame, so the end-to-end delay can be calculated efficiently³. The objective is to keep the delay below ‘ A ’, and preferably below ‘ P ’.

```

while (stream is not completed) {
    if (end-to-end delay  $\leq P$ )
        continue;
    else if ( $P < \text{end-to-end delay} \leq A$ )
        skip 1 frame every  $K$  frames;
    else if ( $A < \text{end-to-end delay}$ )
        skip this frame;
}

```

Figure 2: The frame skipping algorithm.

The frame skipping algorithm is given in Figure 2. It is executed whenever a frame arrives. If the calculated end-to-end delay is shorter than the preferred delay, no action is needed. Otherwise, 1 frame out of K consecutive frames is skipped. If the delay exceeds the acceptable delay bound, the frame will be discarded. Note that for some compression formats, discarding certain types of frames will affect other frames. For example, discarding an I frame in an MPEG stream will affect the P and the B frames in the same group of picture (GOP). Discarding P frames will also affect other P and B frames. In this case, the algorithm in Figure 2 can be modified to check the frame type and discard only non-essential frames (e.g., B frames in MPEG).

Exploiting skips have been studied [13, 14] as a way to reduce the processing demand and alleviate system overload. In the work by West et al. [15], packets can be skipped to reduce the end-to-end delay. However, their algorithm is designed for packet scheduling in network nodes instead of at the end host.

3.2 Rate-controlled Processing with Reservation

In the RCP model, task schedulability guarantees must be upheld using an admission control mechanism. This approach works fine when the WCET of jobs in a task is known. However, it is difficult to estimate the WCET correctly for every frame in a stream. Furthermore, some malicious tasks may demand more capacity than their declared execution times and overload the system. These misbehaving jobs may miss their deadlines as well as causing conforming jobs to miss their deadlines, too. This is unacceptable in an environment where soft and hard real-time tasks are processed together.

---

²Skip means to discard the frame without processing it. In this paper, “skip” and “discard” are used interchangeably.

³Clock synchronization is itself a research issue and is out of the range of this paper.

Algorithm 1 RCS server. Initial values: $k = 0, d_0 = -p_s, overrun_s = 0, \forall k \text{ } budget_k = 0, budget_k^r = 0$.

When job J_i arrives at time t_1 ,

    Enqueue the job in the server's job queue;

if (Only one job in the queue)

        Schedule job J_i to run;

When job J_i is scheduled to run at time t_2 ,

$k = k + 1$;

$a_k = t_2$;

$d_k = \max(a_k + \min(L_s, p_s), d_{k-1} + p_s)$;

    /* The new deadline is one period later than the arrival time or the previous deadline */

if ($a_k \leq d_{k-1}$)

$budget_k = E_s + \min(budget_{k-1}^r, (d_{k-1} - a_k) \cdot E_s / p_s)$;

        /* Inherit the residue budget of the previous job */

else if ($a_k > d_{k-1}$)

$budget_k = E_s \cdot (d_k - a_k) / p_s$;

When server budget is used up,

$k = k + 1$;

$budget_k = E_s$; /* Replenish the budget */

$d_k = d_{k-1} + p_s$; /* Deadline postponed by a period */

$a_k = d_{k-1}$;

$overrun_s = 1$;

When job J_i terminates at time t_3 ,

    Dequeue the job;

if ($overrun_s == 1$)

$d_k = \max(d_k - budget_k^r \cdot p_s / E_s, t_3)$;

        /* Advance deadline to minimize lateness */

$budget_k^r = 0$;

$overrun_s = 0$;

if (job queue not empty)

        Schedule the next job to run;

To address this issue, we propose the rate-controlled server (RCS) to reserve processing capacity for each task and provide isolation among them. Every task has an associated RCS server. All released jobs of a task are put into the waiting queue of the server. We assume frames arrive in order, so the jobs in the same task are serviced by their arrival order. The characteristics of the task defined by (L, E, p)

are inherited by its RCS server. We use E_s and p_s to represent the reserved processor capacity, i.e., the server can run for up to E_s units of time every p_s time units. The utilization of the server is therefore $U_s = E_s/p_s$. Each server maintains a deadline and a budget. The budget denotes the processing time the server is eligible for before the deadline, and is consumed as the server runs. When the budget is used up, the server's budget and deadline are updated according to Algorithm 1.

The underlying system scheduler selects the server with the earliest deadline to run. As the algorithm is work-conserving, the processor will not be idle as long as there are jobs in the waiting queue of any server. Figure 3 shows some snapshots of computation on a server. The shaded segments are the time intervals the server is executing. Each computation segment belongs to a job, while a job may need several segments to complete (e.g., segments 1 and 2 in Figure 3(a) both belong to job J_1). These computation segments are characterized by (a_k, d_k, C_k) , indicating the release time, the deadline and the actual computation time. We define the interval between a_k and d_k as the *active interval* of segment k . Furthermore, we use C_k to denote the actual execution time of the segment. At any time, only one segment can run on the server, and the server deadline is the deadline of that segment. If a job has multiple segments, the deadline of the last segment is taken as the job deadline.

Figure 3: Examples of computation segments of an RCS server.

In a server, we use $budget_k$ and $budget_k^r$ to denote the assigned budget and the residue budget of segment k , respectively. The server budget is the budget of the current segment at any time. If the server budget is used up, the current segment will be terminated and the flag $overrun_s$ is set. A new computation segment will then start; the budget will be replenished by E_s and the server deadline postponed by p_s . In Figure 3(a), as job J_1 overruns, segment 1 terminates and segment 2 is started to handle the rest of the computation. The deadline of segment 2 is postponed to d_2^* and the budget is replenished. If the budget is not used up when an overrun job completes, the deadline of the segment (thus the frame deadline) is advanced as in Algorithm 1, allowing the job to catch up some of the lost time. In Figure 3(a), d_2 is the advanced deadline and the final frame deadline of J_1 . If a job does not overrun and does not use up the budget, its residue budget can be reclaimed by the next job in the task, but the server deadline will still be postponed to control the job completion rate just like an RCP task. In Figure 3(b), when job J_5 completes, job J_6 starts a new segment 7. Inheriting the residue budget of the previous segment, segment 7 can run longer than the reserved processor time E_s , but the deadline

is postponed by p_s (to d_7).

Comparing with other reservation-based algorithms proposed in the literature [5, 6], RCS stands out in the following ways. RCS does not require a priori knowledge of the execution times of the jobs, which are required in CUS [5]. While CBS [6] looks similar to RCS, the underlying design principles are quite different. RCS considers the specified job completion *rate* in addition to the reserved processing *time*. As a multimedia stream is expected to be played back at a constant rate, RCS controls the frame processing rate by spacing consecutive frame deadlines by the specified period. Therefore, when a new job starts execution, the server deadline must be postponed by a period even if the budget of the previous job has not been used up. In comparison, CBS allows a job to assume the same deadline as the preceding completed job, so a constant frame rate is not maintained. For example, if the jobs in a task consume less than the reserved processing time, CBS will process the task faster than the expected pace. Consequently, this task will unnecessarily compete with possibly more urgent jobs in other tasks. Furthermore, the frames completed early have to be stored temporarily, occupying extra buffer space. On the other hand, the RCS framework takes the desirable frame rate into consideration and assigns deadlines which reflects the actual urgency of the frames and effectively maintains the processing rate.

Theorem 5 *An RCS server will not require more than the reserved bandwidth*

$$U_s = E_s/p_s,$$

i.e., the processor demand during any interval $[t_1, t_2]$ will not exceed

$$U_s \cdot (t_2 - t_1) = \frac{E_s}{p_s} \cdot (t_2 - t_1).$$

Proof: The sketch of the proof is given here. For details, please refer to [9].

We modify the release times and deadlines of the computation segments in an RCS as follows:

$$d'_k = d_k - \text{budget}_k^r \cdot p_s / E_s,$$

$$a'_k = \max(d'_{k-1}, a_k),$$

where budget_k^r is the residue budget when segment k has finished execution. The corresponding new segments are non-overlapping and characterized by (a'_k, d'_k, C_k) . Notice that d'_k is no later than the corresponding d_k , and a'_k no earlier than a_k . It can be easily shown that postponing the release times or advancing the deadlines of jobs will not reduce the processor demand or interval utilization in any time interval [9]. Therefore, the modified processor demand and interval utilization are no less than those for the original segments.

According to Algorithm 1, it can be proved that the processor demand C_k during each active interval $[a'_k, d'_k]$ never exceeds $E_s \cdot (d'_k - a'_k)/p_s$. As the intervals are non-overlapping, the demanded bandwidth never exceeds E_s/p_s . Finally, the required bandwidth of the original segments will not exceed E_s/p_s either. $\square$

Corollary 1 *A set of RCS servers ($s = 1 \dots n$) are schedulable if*

$$\sum_{s=1}^n \frac{E_s}{p_s} \leq 1.$$

Proof: The proof directly follows from Theorem 5. If the given condition is satisfied, then the processor demand during any interval $[t_1, t_2]$ is the sum of the processor demands of all RCS servers in the system. The processor demand is at most

$$\sum_{s=1}^n \frac{E_s}{p_s} \cdot (t_2 - t_1) \leq t_2 - t_1.$$

Thus the interval utilization cannot exceed 1. By Theorem 2, the servers are schedulable. $\square$

Theorem 6 *A hard real-time task can be characterized by the minimum inter-arrival time p , the worst-case execution time E and relative deadline (set equal to the minimum inter-arrival time) p . The task is schedulable by EDF if and only if it can be scheduled with an RCS server (p, E, p) , and the RCS server is schedulable.*

Proof: Note that when such a task is scheduled with an RCS server, no overrun will occur and the deadline assigned by the RCS server is exactly the original deadline of the job. Therefore the two methods will build the same schedule and the claim holds. $\square$

4 Performance Evaluation

In this section, we evaluate the performance of the proposed RCP model by simulation and compare it with that of Jeffay's rate-based execution model (RBE) [8]. The effect of the lead time is also examined.

4.1 The RBE Model

RBE [8] models the processing requirements of continuous media streams. It introduces the relative deadline (rd) as the maximum desired time between the release time of a job at the destination host and its completion. A task T is characterized by a three-tuple (rd, E, p) . Deadlines are assigned as follows:

$$d_{i,j} = \begin{cases} r_{i,j} + rd_i & \text{if } j = 1 \\ \max(r_{i,j} + rd_i, d_{i,j-1} + p_i) & \text{if } j > 1 \end{cases}. \quad (2)$$

RCP and RBE differ in the way the relative deadline rd and the lead time L are used in the deadline assignment formulas. RBE always forces a minimum distance (i.e., the relative deadline) between the deadline and the arrival time of a job. If a job arrives later than the expected time, a gap will be produced in the deadline pattern. As a result, RBE tends to preserve the irregular frame arrival pattern in the final frame playback pattern. In RCP, the lead time is used as a buffering mechanism to smooth out the gaps in the deadline pattern. When $L_i = rd_i > p_i$, RCP will assign earlier deadlines to frames arriving late and reduce the number and size of the gaps between consecutive deadlines. However, if $L_i = rd_i \leq p_i$, the two schemes will assign the same deadlines to the frames.

4.2 Performance Metrics

We choose *local delay* and *playback jitter* as the performance metrics to evaluate the algorithm. The former measures the responsiveness of the applications while the latter reflects the smoothness of CM streams.

The local delay (or delay for short) is defined as the time between the assigned frame deadline $d_{i,j}$ and the *expected* frame arrival time $r_{i,j}^*$. The expected frame arrival time of a frame is calculated according to the following formula:

$$r_{i,j}^* = \begin{cases} r_{i,j} & \text{if } j = 1 \\ r_{i,j-1}^* + p_i & \text{if } j > 1 \end{cases}. \quad (3)$$

Assume the frames are released at a constant rate at the source host, the local delay at the destination host differs from the end-to-end delay only by a constant. Specifically, the end-to-end delay of a frame is equal to its local delay plus the network delay of the *first* frame. As the latter is known after the first frame has arrived, the need to estimate the variant network delay for subsequent frames is eliminated.

The playback jitter $Jitter_i$ of a stream can be defined as the maximum absolute difference in the intervals between the assigned frame deadlines. Formally,

$$Jitter_i^{Max-min} = \max_{\forall j \geq 1} (|I_{i,j+1} - I_{i,j}|),$$

where $I_{i,j} = d_{i,j+1} - d_{i,j}$. As the jitter calculated in this way is based on the maximum and minimum intervals between successive frame deadlines, it is also known as the *max-min jitter*. We also define the *mean jitter* as used in the real-time transport protocol (RTP) [16] which reflects longer-term behaviors. The mean jitter is the smoothed absolute value of the difference D in frame spacing at playback compared to the original spacing. For two frames j and $j + 1$ in stream i , $D_{i,j} = (d_{i,j+1} - d_{i,j}) - p_i$. The initial value for the mean jitter is zero. It is continuously updated as each frame is played back as follows:

$\forall j, 1 \leq j < n$,

$$Jitter_i^{Mean} = Jitter_i^{Mean} + (|D_{i,j}| - Jitter_i^{Mean})/16,$$

where n is the number of frames in the stream. This formula is used because it is the optimal first-order estimator and the gain parameter $1/16$ gives a good noise reduction ratio while maintaining a reasonable rate of convergence [16]. The values of the *max-min jitter* and the *mean jitter* are not comparable. However, for the same metric, a smaller jitter value indicates smoother playback.

4.3 Simulation Setup

A simulator has been built to evaluate the performance of the proposed RCP mechanism. The simulated system has a single processor with processing capacity 1 and uses EDF as the scheduling algorithm. We simulated the scenario where multiple video streams were received over the Internet and simultaneously processed in the system.

Figure 4: Average local delays of RBE and RCP.

Figure 5: The Max-min jitter of RBE and RCP.

Due to the difficulty in modelling the arrival patterns, real traces of video streams over the Internet were used as input to the simulator. The transmission/collection was performed between the University of Alberta in Canada and the Hong Kong University of Science and Technology. We used the “foreman” video sequence of qcif format with 400 frames. The sequence was encoded in H.263 format using the tmn codec v1.7 and transmitted by UDP. The frame period was 100ms. The experiment was carried out 24 times, once per hour, which covered the network condition for a whole day.

Then we modified tmndec v1.7 to play back the stream and recorded the processing time of each frame. The computer used was a PIII450 running SuSE Linux 7.1 with kernel 2.4.2. The worst-case processing time of a frame was approximately 6ms while for most frames it was between 3ms to 4ms. As the processing times are more or less linearly dependent on the speed of the processor, they are multiplied by different constants to produce various system load in the following experiments.

4.4 Simulation Results

4.4.1 RCP vs. RBE

We evaluated the performance of RCP and RBE under various frame arrival patterns. The lead time was set to 200ms (twice the frame period) for both models. The base time unit was 1 millisecond. Twenty-four experiments were carried out, each using one of the 24 frame inter-arrival time traces collected.

Figure 6: The Mean jitter of RBE and RCP.

Figure 7: Average local delay of frames by lead times.

Figure 4 shows the average local delay of a frame. The average local delay in the RBE⁴ model varied from 200 to 350ms. Less than half (to be specific, 9 out of 24) of the RBE simulation runs had zero local delay. The proposed RCP model significantly reduced the average local delay: it exceeded the 200ms lead time (note that the delay should be no smaller than the lead time) only in three runs, and was below 260ms in each of those three runs. Due to the granularity in calculating the mean jitter of a stream, we rounded up the jitter to 1.0×10^{-15} if the actual value was smaller. As seen in Figures 5 and 6, the RCP model had the same or smaller jitters as RBE in all the simulation runs. We also varied the lead time between 110ms and 200ms and carried out the same experiments. In all cases, RCP achieved shorter delay, smaller max-min jitter and mean jitter.

4.4.2 Lead Time

In this section, the effect of the lead time in RCP on local delay and playback jitter is examined. We chose the frame arrival traces at times 17, 20 and 6, which had the maximum, minimum and average deviation in the frame inter-arrival times respectively. The frame processing times were multiplied by a factor of 10 to produce a worst-case processing time of 60ms. The frame period p of the stream was kept at 100ms. As the lead time L must be larger than the worst-case processing time E , a set of values from 60 to 300ms in 10ms increments were chosen for the experiments.

⁴Local delay is not defined in RBE, so we evaluated it based on our definition.

Figure 8: Max-min jitter of the stream by lead times.

Figures 7 and 8 show the average local delay and playback jitter of the stream. The delay curves in Figure 7 first increased and became flat when the lead time reached 100ms, which is equal to the value of the frame period. Interestingly, in Figure 8, the three jitter curves started to drop at approximately the same lead time values. As explained in Section 3.1.1, the reduction in jitter was due to using a lead time larger than the frame period (100ms). Furthermore, the jitter curves kept decreasing linearly until they reached zero, when the corresponding delay curves in Figure 7 started to increase again after a horizontal section. The corresponding lead time values for these turning points are related to the *maximum frame arrival delay*, i.e., the maximum distance between the frame arrival and its expected arrival time. Note that the maximum frame arrival delays of the max, min and average patterns were 153ms, 19ms and 25ms, respectively, while the corresponding lead times were 253ms, 119ms and 125ms, i.e., the sum of the frame period and the maximum frame arrival delay. Several conclusions can be drawn from these observations: First, increasing the lead time beyond the frame period can reduce the jitter in the deadline pattern, but the local delay will increase as well. Second, a lead time smaller than the frame period cannot reduce jitter but will produce a shorter local delay. Third, there is no benefit in increasing the lead time beyond the maximum frame arrival delay.

5 Conclusion

We have presented a rate-controlled processing (RCP) model for scheduling networked multimedia applications at the destination hosts. The model is especially useful for interactive applications with stringent end-to-end delay requirements. The frame processing rate is regulated by assigning frame deadlines intended to compensate for the irregular frame arrival pattern. Furthermore, using the concept of frame deadlines eliminates the time gaps between the frame processing deadline and frame playback time.

The schedulability conditions of RCP tasks have been derived. Simulation results have shown that RCP achieves lower delay and jitter compared to the rate-based execution (RBE) model. Choosing a proper lead time directly affects the delay and jitter of the stream. To further reduce end-to-end delay, RCP with skips has been proposed for multimedia applications in which some amount of frame loss is

tolerable. Some frames can be skipped according to the delay requirements and the skip constraint. In systems where hard and soft real-time tasks are processed together, we have proposed the rate-controlled server (RCS) to prevent misbehaving tasks from interfering with the other tasks and protect hard real-time tasks so they can meet their deadlines.

References

- [1] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 83(10):1374–1399, 1995.
- [2] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
- [3] I. Stoica, H. A. Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke and C. G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 288–299, 1996.
- [4] P. Goyal, X. Guo and H. M. Vin. A hierarchical CPU scheduler for multimedia operating systems. *ACM Operating Systems Review*, 30(special issue):107–121, 1996.
- [5] Z. Deng, J. Liu, and J. Sun. A scheme for scheduling hard real-time applications in open system environment. In *Proceedings of the Ninth Euromicro Workshop on Real-Time Systems*, pages 191–199, 1997.
- [6] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 4–13, 1998.
- [7] G. Lipari and S. Baruah. A hierarchical extension to the constant bandwidth server framework. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 26 –35, 2001.
- [8] K. Jeffay and S. Goddard. A theory of rate-based execution. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 304–314, 1999.
- [9] F. Zhang and S. Chanson. Rate-controlled scheduling for networked multimedia applications. Technical Report HKUST-CS01-07, Department of Computer Science, Hong Kong University of Science and Technology, September 2001.
- [10] M. L. Dertouzos. Control robotics: The procedural control of physical processes. In *Proceedings of the IFIP Congress*, pages 807–813, 1974.
- [11] M. Spuri. *Earliest Deadline Scheduling in Real-Time Systems*. PhD thesis, Scuola Superiore S. Anna, Pisa, Italy, 1995.
- [12] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 182–190, 1990.
- [13] P. Ramanathan. Overload management in real-time control applications using (m,k)-firm guarantee. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):549 –559, 1999.
- [14] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 110–117, 1995.

- [15] R. West and C. Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort packet streams. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 239–248, 2000.
- [16] Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC 1889: RTP: A transport protocol for real-time applications, January 1996.