Skip-Over: Algorithms and Complexity for Overloaded Systems that Allow Skips*

Gilad Koren
Computer Science Institute
Bar-Ilan University
Ramat-Gan, 52900 Israel

Dennis Shasha
Courant Institute
New York University
New York, NY 10012

January 10, 1996

Abstract

In applications ranging from video reception to telecommunications and packet communication to aircraft control, tasks enter periodically and have fixed response time constraints, but missing a deadline is acceptable, provided most deadlines are met. We call such tasks "occasionally skippable". We look at the problem of uniprocessor scheduling of occasionally skippable periodic tasks in an environment having periodic tasks. We show that making optimal use of skips is NP-hard. We then look at two algorithms called Skip-Over Algorithms (one a variant of earliest deadline first and one of rate monotonic scheduling) that exploit skips. We give schedulability bounds for both.

1 Introduction

1.1 Basic Assumptions and Definitions

We consider a uni-processor system in which preemption is possible at any time and costs nothing. All tasks are periodic but they may enter the system at any time. A task is characterized by its computation requirements and period; the deadline of a task equals its period. Tasks are assumed to be independent, i.e., there are no precedence constraints among different tasks. A task t is divided into instances where each instance occurs during a single period of the task. Every instance of a task can be *red* or *blue*. A red task instance must complete before its deadline; a blue task instance can be aborted at any time. We borrow this colorful terminology from the Swedish VIA project [8]. A blue task instance may complete by its deadline or miss its deadline. When a task misses its deadline we say that the task (or deadline) instance was *skipped.* We characterize the possible skips of a task by its *skip parameter*, $2 \leq s \leq \infty$, which gives the tolerance of this task to missing deadlines. The distance between two consecutive skips must be at least s periods. That is, after missing a deadline at least $s-1$ task instances must meet their deadlines. The fact that $s \geq 2$ implies that if a blue task instance missed its deadline then the next occurrence of the same task must be red. When $s = \infty$ no skips are allowed. One can view the skip parameter as a *Quality of Service (QOS)* metric. (The higher s, the better the quality of service.)

1.2 Examples to Train Your Intuition

Allowing skips may permit us to schedule systems that might otherwise be overloaded.

- *Example 1.* We have a set of 11 tasks all having period 1, computation time 0.1, and skip factor 10. All 11 tasks can be accommodated provided that at every period of length 1, one task instance can be skipped.

- *Example 2.* We have two tasks: t_1 has period 1, computation time 1 and skip factor 10; t_2 has period 1, computation time 0.05 and skip factor infinity. This system can't be scheduled because t_1 can't give time to t_2 frequently enough even though a naive calculation of utilization, taking skips into account, $(0.9 + 0.05)$ is less than 1.

- *Example 3.* We have two tasks: t_1 has period 1, computation time 1, and skip factor 10; t_2 has

*This paper appeared in the proceedings of the 16th Real-Time Systems Symposium, Pisa Italy, December 1995.
Supported by U.S. Office of Naval Research grants #N00014-91-J-1472 and #N00014-92-J-1719, U.S. National Science Foundation grant #CCR-9103953.
NSF grant #IRI-9224601
Authors' e-mail addresses :
koren@bimacs.cs.biu.ac.il, and shasha@cs.nyu.edu
Author fax: 212-995-4123 (but we hate faxes)

period 15, computation time 1 and skip factor infinity. This system can be scheduled because every period of t_2 overlaps at least one skippable task instance of t_1. Note that the utilization here $(0.9 + 1/15)$ is actually higher than in example 2 above.

1.3 Alternative Models

Our model captures the situation of periodic radar signals entering a system where capturing every other one makes sense. It also captures the video display problem where we are willing to skip an occasional frame but frame arrivals are periodic. We rejected the following possible models because we considered them to be too permissive. Since they allow more freedom to the scheduler than our current model, however, our algorithms will work for those models.

- *Stale Skips*: If a task skips the current period while working on instance i, it can complete i by the deadline of the following period and count that as a single skip. In our model, an instance which misses its period is not worth completing.
- *Early Release*: If a task instance is skipped, the following red task may be released before the end of the period. In our model, skipping a task instance has no effect on the release time of the next instance.
- *Statistical Model*: Another possible model is a statistical one which says that at least some fraction of deadlines must be met during every some basic period.

2 The Feasibility Problem for Skippable Tasks

The feasibility problem amounts to the following question. Can we find a collection of deadlines to be skipped such that no skip parameter is violated and the remaining tasks can be scheduled to complete before their respective deadlines? In the basic periodic model, in which all task instances are red ($s = \infty$ for all tasks) , it was shown [15] that a task set $\{T_i; 1 \le i \le n\}$ is schedulable[1] if and only if its *cumulative processor utilization* (ignoring skips) is no greater than 1. I.e.,

$$\sum_{i=1}^{n} \frac{c_i}{p_i} \le 1$$

[1] Also, if a set is feasible, it can be scheduled using the *Earliest-Deadline-First* Algorithm (EDF) [15].

Task	T_1	T_2
Period	100	50
Computation	$100 - 100\epsilon$	100ϵ
Skip	2	2

Table 1: The tasks for example 2.1.

Task	T_1	T_2
Period	100	$100/k$
Computation	$100 - \epsilon$	ϵ

Table 2: The tasks for example 2.2.

This result motivated us to look for a similar necessary and sufficient (or at least sufficient) condition for schedulability. Unfortunately no better sufficient condition, based solely on cumulative processor utilization, exists. To see that let us concentrate on the special case where $s = 2$ uniformly for all tasks. That is, every other task instance can be skipped. Consider the following inequality:

$$\sum_{i=1}^{n} \frac{c_i}{p_i} \le \mathcal{U} \tag{1}$$

It is clear that for $\mathcal{U} = 2$ equation (1) is a necessary condition for schedulability because of the constraint on s. Similarly, $\mathcal{U} = 1$ is a sufficient condition, because it implies that the task set is schedulable even when no deadline is skipped. We tried to find a sufficient condition of the above form with $\mathcal{U} > 1$. Unfortunately this is impossible as the following example demonstrates:

Example 2.1 Consider the tasks of table 1. The processor utilization of this set (ignoring skips) is $U = 1 + \epsilon$. It is schedulable (by for example, scheduling the first occurrence of each task and then skipping any other deadline). But enlarging the computation time requirement of any of the tasks will render the task set infeasible. Since this is true for all $\epsilon > 0$ the claim is proved.

Surprisingly, the same observation holds even when consecutive task instances may be skipped. Consider for example the following example in which up to $k \ge 1$ consecutive instances of each task may be skipped.

Example 2.2 Consider the tasks of table 2. The processor utilization of this set is $U = 1 + \frac{\epsilon(k-1)}{100}$ (which can be as close to 1 as we want). It is schedulable (by for example, scheduling every kth occurrence of T_2 and all occurrences of T_1). However, any increase in the computation of T_2 renders the system unschedulable.

We do have the following necessary condition for schedulability

Lemma 2.1 *Given a set* $\Gamma = \{T_i(p_i, c_i, s_i)\}_{i=1}^{n}$ *of periodic tasks that allow skips, then,*

$$\sum_{i=1}^{n} \frac{c_i(s_i - 1)}{p_i s_i} \leq 1 \qquad (2)$$

is a necessary condition for the feasibility of Γ*, because that sum expresses the utilization based on the computation that must take place.*

3 Skip-Over for Earliest Deadline First

3.1 Processor Demand Criteria

After realizing that the parameter of cumulative processor utilization was not itself sufficient for distinguishing between feasible and infeasible tasks sets, we turned to another known form of schedulability test for EDF: the *processor demand* criteria. Jeffay and Stone [7] showed that a set of periodic tasks will be schedulable if and only if for all $L \geq 0$,

$$L \geq \sum_{i=1}^{n} \left\lfloor \frac{L}{p_i} \right\rfloor c_i$$

Similar bounds can be found in our case.

Lemma 3.1 *Given a set* $\Gamma = \{T_i(p_i, c_i, s_i)\}_{i=1}^{n}$ *of periodic tasks that allow skips, then,*

$$L \geq \sum_{i=1}^{n} D(i, [0, L]) \qquad \text{for all } L \geq 0 \qquad (3)$$

Where,

$$D(i, [0, L]) = (\lfloor \frac{L}{p_i} \rfloor - \lfloor \frac{L}{p_i s_i} \rfloor) c_i \qquad (4)$$

is a sufficient condition for the feasibility of Γ.

PROOF.

First assume that all tasks are released at time 0. Consider the schedule in which for T_i, every s_i'th deadline is skipped (that is, exactly the s_i'th, $2s_i$'th, $3s_i$'th deadlines are skipped). All non-skipped instances are scheduled according to EDF. Denote by $D(i, [t_1, t_2])$ the computation demand of T_i during the interval $[t_1, t_2]$ under the above schedule. That is the total computation requirement of all tasks of type T_i that were released and must complete within the interval $[t_1, t_2]$. For an interval $[0, L]$ the value of D is given by equation 4 above. Condition 3 above is sufficient since it actually shows that Γ is schedulable using the schedule described above. Suppose Γ is not schedulable, that is at some time d some task instance misses its deadline. Let $t_a \geq 0$ be the last time (prior to d) that the processor was left idle by the above scheduling algorithm (let $t_a = 0$ if there is no such point). Let $t_b \geq 0$ be the last time (prior to d) that the processor was busy executing any task instance with deadline after d (let $t_b = 0$ if there is no such point). Let $t = \max\{t_a, t_b\}$. The skip pattern of the algorithm insures that $D(i, [t, d]) \leq D(i, [0, d - t])$. The time t has the property that only task instances that were released after t with deadline on or before d are executed during $[t, d]$. There is no idle time in $[t, d]$. Hence, the optimality of EDF [5] and the fact that a deadline was missed means that the computation demand during this interval is greater than the interval length. That is,

$$d - t < \sum_{i=1}^{n} D(i, [t, d]) \qquad (5)$$

Hence, we get

$$d - t < \sum_{i=1}^{n} D(i, [t, d]) \leq \sum_{i=1}^{n} D(i, [0, t - d]) \qquad (6)$$

This proves the claim for the case that all tasks start at time 0. The case in which tasks may arrive at any time is similar. Here, as above, the scheduler waits $s_i - 1$ periods from T_i's first release before the first skip and then skip in gaps of s_i periods. If an overload occurred, choose d and t as above. Let D' denote the processor demand function for this system. Observe that, for every t and d, the skip pattern followed by the above scheduling algorithm satisfies

$$D'(i, [t, d]) \leq D(i, [0, d - t])$$

. Hence, we get

$$d - t < \sum_{i=1}^{n} D'(i, [t, d]) \leq \sum_{i=1}^{n} D(i, [0, t - d]) \qquad (7)$$

□

It is enough to check inequality 3 above for points L that are periods' end points. Moreover there is no need to check any point L beyond[2] $P = lcm(p_1, p_2, \cdots, p_n)$. Note that the value of P is not the expected $lcm(p_1 s_1, p_2 s_2, ...)$. Checking only for $L < P$ is enough because the algorithm skips at the *end* of $p_i s_i$ intervals. This means that if the scheduler succeeded for $[0, P]$ it will succeed forever.

[2] $lcm(p_1, p_2, \cdots, p_n)$ denotes the least common multiple of $p_1, p_2, \cdots p_n$.

3.2 Feasibility With Skipping is NP-Hard

When the first task instance of some task is released is it red or blue? For a task with skip factor s, which is its first blue instance? The first, second or maybe only the s'th. If the first blue instance of any task is the s'th we say that the system is *deeply red*. Of course when a system is schedulable assuming it is deeply red it is also schedulable without this assumption. First, we study the question of how hard it is to determine the best schedule off-line in the general case that tasks may be blue when they enter.

Theorem 3.2 *Determining whether a set of periodic occasionally skippable tasks is schedulable is NP-hard.*

The reduction is from the *Partition Problem* [6]:

- **instance:** Finite set A and a (positive integer) size $s(a)$ for each $a \in A$.
- **question:** Is there a subset $A^{'} \subseteq A$ such that $\sum_{a \in A^{'}} s(a) = \sum_{a \in A - A^{'}} s(a)$?

Given an instance of the partition problem, we can construct (in polynomial time) a corresponding set of occasionally skippable tasks such that the partition is possible if and only if the corresponding set is feasible. The corresponding task set $\mathcal{A}$ is defined as follows: The number of tasks equals the number of elements in A, for each $a \in A$ there is a task in $\mathcal{A}$ with computation time $s(a)$. All tasks have the same period which equals $\sum_{a \in A} \frac{s(a)}{2}$ and a common skipping parameter 2. In the specific case of $\mathcal{A}$ above, the scheduling problem is reduced to the question of which tasks will skip every *odd* deadline and which will skip every *even* deadline. This partitioning of the tasks is possible if and only if it is possible to partition the original set A.

Remark 3.1 Since the partition problem is NP-hard in the weak sense we have proved here only that the feasibility problem is NP-Hard in the weak sense.

Remark 3.2 We assume here that when tasks enter, their task instances are blue to begin with. We conjecture that the offline scheduling problem is NP-hard in the deeply red model, but we have no proof. The RTO scheduling algorithm presented below is optimal in the deeply red model (lemma 4.1 below). Hence, it can be used as an offline feasibility test. Its complexity however depends on the values of task periods and may be exponential.

The complexity of the feasibility problem was studied by [14, 13, 1]. They look at scheduling a periodic task set with arbitrary deadlines and arbitrary initial offsets. In addition to its period (p) and computation requirement (c) a task is characterized by its deadline ($c < d < p$) and first release point ($r \geq 0$). Given a task set Γ with skips, consider a modified task set Γ' in which every task $T = (c, p, s)$ is replaced by $s - 1$ tasks all with period ps, deadline p and computation requirement c and initial offsets $0, p, 2p, \cdots, (s-2)p$. The optimality of RTO in the deeply red case implies that Γ is schedulable if and only if Γ' is schedulable. Baruah and Rosier [1] showed that the general problem of deciding schedulability of such task sets is co-NP-complete in the strong sense.

4 Scheduling Algorithms

In the previous section we studied the problem of checking the feasibility of a system. In this section we would like to present some on-line scheduling algorithms for such systems.

4.1 Red Tasks Only Algorithm

The first algorithm is the *Red Tasks Only (RTO)* algorithm. This algorithm is a lazy algorithm in the sense that it never attempts to schedule a blue task (i.e., it never works unless it absolutely has to). The red tasks are scheduled according to EDF. In the deeply red model this algorithm is optimal, I.e.. all feasible sets will be schedulable using RTO.

Lemma 4.1 *In the deeply red model RTO is optimal.*

PROOF.
In this special case the sufficient condition (equation 3) becomes also a necessary condition. The task of a scheduler is to choose which tasks instances to skip. Once this is done it is clear from the optimality of EDF [5] that scheduling the un-skipped tasks is done best by EDF. The skipping policy of a scheduler determines its computation demand function. Note, that in the deeply red case $D(i, [0, L])$ (of lemma 3.1) is the minimal computation demand over the interval $[0, L]$ among all valid skiping policies. This holds because, in the algorithm described in the lemma, the number of skipped tasks is the largest possible for all task types. In the deeply red case that algorithm behaves exactly like RTO. Hence, if for some $L > 0$,

$$L < \sum_{i=1}^{n} D(i, [0, L])$$

Then for all possible skipping policies, the computation demand over $[0, L]$ will exceed the available computation time L. $\square$

Task	T_1	T_2
Period	10	5
Computation	7	3
Skip Parameter	2	2

Table 3: A task set schedulable by RTO but not by EDF

Task	T_1	T_2
Period	6	4
Computation	4	3
Skip Parameter	2	2

Table 4: The tasks for example 4.1.

RTO can schedule task sets that an EDF that doesn't use skips will fail to schedule. (see for example the tasks of table 3). We saw that RTO is optimal in the deeply red case, but RTO is not optimal in general as the following example demonstrates:

Example 4.1 Consider the tasks of table 4. If both first releases are red then the system is not feasible: the first deadline of T_1 (at time 6) will be missed, However, in all other cases the system is feasible but RTO will fail to schedule the tasks in such a way that they meet their deadlines.

- If both first releases are blue than RTO will schedule only the even occurrences of each task type. The T_1 task released at time 18 and the T_2 task released at 20 have both deadline at time 24 but their aggregated computation time (= 7) is greater than 6 causing a red deadline miss.
- If the first occurrence of T_1 is blue and the first occurrence of T_2 is red. RTO will schedule every odd occurrence of T_2 and every even occurrence of T_1. One can verify that the third occurrence of T_2 will miss its deadline.
- A similar phenomena happens when the first occurrence of T_1 is red (and the first T_2 is blue), in this case both the second occurrence of T_1 and the third occurrence of T_2 are released at time 12 causing an overload.

In all the above three cases the tasks would meet their deadlines if we scheduled every even occurrence of T_1 (when the first occurrence of T_1 is blue) and skipped every third occurrence of T_2 (i.e., scheduled the first and the second occurrences and then the forth and the 5th etc.).

RTO skips deadlines in a "regular fashion", that is the distance between every two skips is exactly s periods. RTO sets the first skip to be at the first blue period. Is it possible that delaying the first skip (and then continuing in a regular fashion) will lead to a feasible schedule? That is can we reduce the problem of choosing which deadlines to skip to choosing the first deadline (per task type) to be skipped? The answer is "No!" as the previous example demonstrates. The system is feasible but not by any regular scheduler.

4.2 Blue When Possible Algorithm

Another more flexible algorithm is the *Blue When Possible (BWP)* algorithm. Its philosophy is to schedule blue tasks whenever this does not prevent any red task from completing, thus putting idle time to good use. (Think of this algorithm when you turn on the Super Bowl instead of writing your paper for RTSS 96.)

The algorithm: schedule red tasks according to EDF. If there are no ready red tasks, then dispatch a blue task. If there are more then one such blue tasks which one to dispatch? We can suggest several heuristics:

- any one
- the blue task with latest deadline
- the blue task with earliest deadline
- some lookahead (that is add a blue task) and see that it does not introduce an overload for some future time[3]
- the blue task with the following property: *the deadline of its next red occurrence (yet to be released) is the earliest*

It is easy to find examples in which BWP improves on RTO. For example the task set of example 4.1 is schedulable by BWP but not by RTO. But all we have been able to show so far is that it is no worse.

Theorem 4.2 *For a given task set, if it is feasible assuming all tasks released at time 0 and are deeply-red it is schedulable using BWF.*

PROOF.

Lemma 4.1 implies that if a system is feasible assuming all tasks starts at time 0 and are deeply red, it is schedulable using RTO.

[3] A feasble schedule for this future period must then be found. The length of the schedule should depend on the blue task's period.

We continue the proof by induction: If the system was initially schedule by RTO, it will be so after the first dispatch of blue task, and then after the second and so forth.

The argument stands on the observation that the worst case (from point of view of RTO) is that all tasks are released at the same time and they are all deeply red. We know that the system is schedulable under these worst case assumptions.

Suppose at time t idle time occurs and BWP schedules a blue instance of T_1 (the next red deadline of T_1 becomes blue). Let $t_2, \cdots, t_n$ denote the (future) release times of the next instances of $T_2, \cdots, T_n$ respectively. If the schedule is to continue according to RTO, it is as if RTO is to schedule a task set with initial release times at $t, t_2, \cdots, t_n$ (deeply red or not). RTO is guaranteed to succeed (because it does so even in the worst case).

□

An avenue to explore is to consider scheduling a blue task every time it is released (i.e, not waiting for the time in which the idle time begins).

5 Rate Monotonic Red Tasks Only

Rate-Monotonic RTO (RM-RTO), like RTO, is a lazy algorithm that schedules red tasks only, but in this case the underlying scheduling algorithm is a fixed priority scheduler where priorities are given according to the Rate-Monotonic [15] priority scheme. Given a task set, we are interested to know whether it is schedulable using RM-RTO under all possible task phasings. The *critical instance test* [15] for testing schedulablity of a task set is applicable here as well. That is, a task set is schedulable using the RTO-RM algorithm if the first job of each task can meet its deadline when it is initiated at a critical instant. (A critical instance occurs when all tasks are simultaneously initiated and all tasks are deeply red.) This holds because the worst case response time is obtained at a critical instance. The characterization for RM schedulablity given by Lehoczky, Sha, and Ding [12] can be adjusted to our model, in the following way: Suppose we are given a set of n periodic tasks with skips $T_1, T_2, \cdots, T_n$, sorted by their periods [4] so that $p_1 \le p_2 \le \cdots \le p_n$. The expression

$$W_i(t) = \sum_{j=1}^{i} c_j \cdot (\lceil t/p_j \rceil - \lfloor \lceil t/p_j \rceil / s_j \rfloor)$$

gives the cumulative processor demand made by $T_1, T_2, \cdots, T_i$ when all tasks are first released at time 0. Define,

$$\begin{aligned} L_i(t) &= W_i(t)/t \\ L_i &= \min_{0 \le t \le p_i} L_i(t) \\ L &= \max_{1 \le i \le n} L_i \end{aligned}$$

Theorem 1 of Lehoczky, Sha, and Ding [12] is aplicable in this case as well.

Theorem 5.1 *A set of periodic tasks with skips can be scheduled using the RM-RTO algorithm if and only if* $L \le 1$

PROOF.

(following Luhoczky, Sha, and Ding [12] nearly verbatim)

Task T_i will be preempted only by higher priority tasks and will preempt all lower priority tasks. Thus only higher priority tasks $T_1, \cdots, T_{i-1}$ have to be considered in determining whether T_i can be scheduled.

T_i will complete at $0 < t \le p_i$ if and only if T_i and all higher priority tasks released before t could complete on or before t. As a matter of fact, since there can be no idle time before t, the total computation requirement of these tasks (given by $W(t)$) must equal t. Note that, $W_i(s) > s$ for $0 < s < t$, hence T_i completes at the smallest t such that $W_i(t) = t$. That is T_i completes on time if an only if $L_i \le 1$. It follows that all the tasks are schedulable if and only if $L_i \le 1$ for all i, hence $L \le 1$

□

Liu and Layland proved that any set of n tasks whose utilization $U = \sum_1^n c_i/p_i$, satisfies:

$$U \le U(n) = n(2^{1/n} - 1)$$

is schedulable using RM. Suppose $T_1, T_2, \cdots, T_n$ is not schedulable using RM-RTO. but the subset $T_1, T_2, \cdots, T_{n-1}$ is schedulable using RM-RTO. Suppose we modify the above task set so that skips are not allowed but the computation requirement of each task among $T_1, T_2, \cdots, T_{n-1}$ is reduced so that its utilization is unchanged. That is, task T_i has a modified computation requirement of

$$c_i^* = \frac{s_i - 1}{s_i} c_i$$

This would mean that from time 0 to p_n the time available for T_n's execution is increased by some $\alpha \ge 0$. That is, if the execution requirement of T_n is increased

[4] Hence the tasks are also sorted by their RM priorities.

Task	T_1	T_2	T_3
Period	6	7	19
Computation	1	4	5
Skip Parameter	2	2	2

Table 5: The tasks for example 5.1.

by α the resulting set of periodic tasks is guaranteed to be unschedulable (using RM). Applying the Liu and Layland utilization bound implies:

$$U = \sum_{1}^{n-1} \frac{c_i^*}{p_i} + \frac{c_n}{p_n} + \frac{\alpha}{p_n} > U(n) \quad (8)$$

We would like to estimate α. The contribution of task T_i to α is at most $c_i^* = \frac{s_i - 1}{s_i} c_i$. Hence, $\alpha \leq \sum_{i=1}^{n-1} c_i^* = \sum_{i=1}^{n-1} \frac{s_i-1}{s_i} c_i$, which applied to 8 yields,

$$U_n \stackrel{\text{def}}{=} \sum_{1}^{n-1} \frac{c_i^*}{p_i} + \frac{c_n}{p_n} + \sum_{i=1}^{n-1} \frac{c_i^*}{p_n} > U(n)$$

Theorem 5.2 *If for all $1 \leq i \leq n$ $U_i \leq U(i)$, then the the task set is schedulable using RM-RTO.*

PROOF.

Suppose the set is not RM-RTO schedulable. Let i_0 be the first i such that $T_1, T_2, \cdots, T_{i_0}$ is not schedulable using RM-RTO, but $T_1, T_2, \cdots, T_{i_0 - 1}$ is schedulable using RM-RTO. Then the discussion above implies that $U_i > U(i)$. Contradiction. $\square$

Corollary 5.3 *If for all $1 \leq i \leq n$ $U_i \leq 69\%$ then the the task set consisting of $T_1, \cdots, T_n$ is schedulable using RM-RTO.*

PROOF.

True since $U(n) > 69\%$ for all n. $\square$

The following example demonstrates the usage of the utilization bounds found above.

Example 5.1 Consider the tasks of table 5. The full-utilization (i.e., the utilization ignoring skips) of this task set is above 1 hence it is unschedulable w/o skips. However theorem 5.2 shows that the task set is schedulable using RM-RTO:

$i = 2$:

$U_2 = \frac{1}{12} + \frac{4}{7} + \frac{1}{2*7} = 0.73 \leq u(2) = 0.82$

$i = 3$:

$U_3 = \frac{1}{12} + \frac{4}{14} + \frac{5}{19} + \frac{5}{38} = 0.764 \leq u(3) = 0.779$

Lehoczky [9] studies fixed priority scheduling of periodic tasks in which all deadlines are deferred (or advanced) by some fixed factor $\Delta > 0$; the deadline of a task is Δ times its period. The results described there for $\frac{1}{2} \geq \Delta > 0$ are applicable to our model in the special case that *all* tasks share the *same* skip factor s ($\Delta = 1/s$). The utilization bound quoted there (due to [11, 16]) is of $1/s$ for all $s \geq 2$. One can see that even in this special case while the task set of example 5.1 fails the test of [11, 16] (because its utilization is greater than 1/2) it is still proven schedulable using the bound derived above.

6 Conclusion

Like telemarketing solicitations and political speeches, repeated signals arriving to a controller contain a great deal of redundancy. It is therefore sometimes acceptable to skip some of these signals. The question is: what can one gain by skipping them? This paper has presented algorithms that can take advantage of skipped task instances to schedule even overloaded systems under a fairly conservative model (no two skips in a row and a task instance that meets the current deadline must be the instance of the current period). The paper has also shown that determining the best task instances to skip is NP-hard. Since the Skip-Over algorithms allow idle time, we can use all the standard tricks for fitting sporadic tasks into that idle time, e.g., [17, 4, 10] and [2] for the *Earliest-Deadline-First* Algorithm scheduling. Future theoretical work includes:

1. In the fixed priority model, we must determine how best to use idle time. For example, is there a good variant of Blue When Possible for that model? Or maybe one should give different priorities to blue and red instances of a task, so that blue tasks will execute as long as they do not harm red tasks. (for example use the Dual Priority Scheme [3])

2. Find an algorithm to deal with tasks whose skip parameters change over time.

Exploiting allowable skips is a tantalizing possibility for systems that experience overload. The two simple algorithms we discuss here are just the first of many that wait to be discovered and evaluated.

7 Acknowledgments

We would like to thank Doug Jensen, Doug Locke, and H. R. Callison for suggesting variants of this problem over the phone, over dinner, and in a van.

References

[1] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *The Journal of Real-Time Systems*, 2:301–324, 1990.

[2] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10):466–473, 1989.

[3] R. Davis. Dual priority scheduling. Computer science department technical report, York University, UK, 1995. And in this proceedings.

[4] R. Davis, K. Tindell, and A. Burns. Scheduling slack time in fixed priority preemptive systems. In *Proceedings of the 14th Real-Time Systems Symposium*, pages 222–231, Raleigh-Durham, NC, Dec. 1993. IEEE.

[5] M. L. Dertouzos. Control robotics: The procedural control of physical processes. In *Proceedings of the 1974 IFIP Congress*, pages 807–813, 1974.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: a guide to the theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[7] K. Jeffay and D. Stone. Accounting for interrupt handling costs in dynamic priority task systems. In *Proceedings of the 14th Real-Time Systems Symposium*, pages 212–221, Raleigh-Durham, NC, Dec. 1993. IEEE.

[8] H. Lawson, M. Lindgren, M. Strömberg, T. Lundqvist, K.-L. Lundbäck, L.-Å. Johansson, J. Torin, P. Gunningberg, and H. Hansson. BASEMENT: A Distributed Real-Time Architecture for Safety Critical Applications. In J. Wikander, editor, *Proc. SNART Symp. on Real-Time systems.* DAMEK, Royal Institute of Technology, Stockholm, 1993.

[9] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 201–209. IEEE, Dec. 1990.

[10] J. P. Lehoczky and S. R. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Proceedings of the 13th Real-Time Systems Symposium*, pages 110–123, Phoenix, Arizona, Dec. 1992. IEEE.

[11] J. P. Lehoczky and L. Sha. Performance of real-time bus scheduling algorithms. *ACM Performance Evaluation Review*, 14, 1986.

[12] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm – exact characterization and average case behavior. In *Proceedings of the 10th Real-Time Systems Symposium*, pages 166–171. IEEE, Dec. 1989.

[13] J. Y.-T. Leung. A new algorithm for scheduling periodic, real-time tasks. *Algorithmica*, 3:209–219, 1989.

[14] J. Y.-T. Leung and M. L. Merrill. A note on preemptive scheduling of periodic. real-time tasks. *Information Processing Letters*, 11(3):115–118, 1980.

[15] C. L. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-tim environment. *Journal of the ACM*, 20(1):46–61, 1973.

[16] D.-T. Peng and K. Shin. A new performance measure for scheduling independent real-time tasks. Technical report, Real-Time Computing Laboratory, University of Michigan, 1989.

[17] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic scheduling for hard real-time system. *The Journal of Real-Time Systems*, 1:27–60, 1989.