Exploiting Skips In Periodic Tasks For Enhancing Aperiodic Responsiveness

Marco Caccamo and Giorgio Buttazzo
Scuola Superiore S. Anna, Pisa, Italy
marea@tirreno.it, giorgio@sssup.it

Abstract

In certain real-time applications, ranging from multimedia to telecommunication systems, timing constraints can be more flexible than scheduling theory usually permits. For example, in video reception, missing a deadline is acceptable, provided that most deadlines are met. In this paper, we deal with the problem of scheduling hybrid sets of tasks, consisting of firm periodic tasks (i.e., tasks with deadlines which can occasionally skip one instance) and soft aperiodic requests, which have to be served as soon as possible to minimize their average response time. We propose and analyze an algorithm, based on a variant of Earliest Deadline First scheduling, which exploits skips to enhance the response time of aperiodic requests. Schedulability bounds are also derived to perform off-line analysis.

1 Introduction

Many real-time applications require periodic activities that have to be cyclically executed at fixed rates and within specific deadlines. Typically, each periodic instance is assigned a relative deadline equal to the task period and is treated as a hard job. Thus, a periodic task is executed only if all its instances are guaranteed to complete within their deadlines. Schedulability analysis of periodic task sets can easily be performed both under fixed and dynamic priority assignments. In particular, a lot of work has been done for the Rate Monotonic (RM) and the Earliest Deadline First (EDF) algorithms [9]. Schedulability analysis has also been extended for the case in which tasks use shared resources [11, 1, 2] or run in the presence of aperiodic activities, under fixed priority scheduling [7, 12, 8] and in dynamic priority systems as well [13, 3, 15].

All these papers assume that periodic tasks are hard, that is, all instances of a task have to be executed within their deadlines, otherwise the task is not started at all. This model is suitable for critical control applications in which missing a deadline may cause catastrophic consequences on the controlled system, but it can be too restrictive for other applications. For example, in multimedia communication systems, skipping a video frame once in a while decreases the quality of service but certainly does not cause any damage to the system. Even in more critical control applications, hard tasks usually coexist with soft and non real-time activities, which need to be treated in a different manner in order to optimize the available resources.

Permitting skips in soft periodic tasks increases system flexibility, since it allows to make a better use of resources and schedule systems that would otherwise be overloaded. Consider for example two periodic tasks, τ_1 and τ_2, with periods $p_1 = 10$, $p_2 = 100$, and execution times $C_1 = 5$ and $C_2 = 55$, where τ_1 can skip an instance every 10 periods, whereas τ_2 is hard (i.e., no instances can be skipped). Clearly, the two tasks cannot be both scheduled as hard tasks, because the processor utilization factor is $U = 5/10 + 55/100 = 1.05 > 1$. However, if τ_1 is permitted to skip one instance every 10 periods, the spare time can be used to accommodate the execution of τ_2.

The general problem of scheduling periodic tasks that allow occasional skips has not received much attention in the real-time literature. Hamdaoui and Ramanathan [4] proposed a general framework called (m, k) model, for dealing with periodic tasks having firm deadlines. However, this method is based on a heuristic priority assignment and no exact schedulability analysis is performed. Another general model for increasing flexibility of periodic scheduling has been proposed by Mok and Chen [10]. Although skippable tasks could be handled with this model, this technique can only be applied to fixed priority environment, in the absence of aperiodic tasks.

The most significant approach on skippable periodic tasks is the one carried out by Koren and Shasha [6]. They showed that making optimal use of skips is NP-hard and proposed two algorithms, called Skip-Over Algorithms (one a variant of rate monotonic scheduling and one of earliest deadline first) that exploit skips to increase the feasible periodic load and schedule slightly overloaded systems.

In this paper, we address the problem of scheduling hybrid task sets, consisting of firm (i.e., skippable) periodic tasks and soft aperiodic requests. In such an environment, we propose an algorithm that uses the spare time saved by the skipped instances to enhance the response time of aperiodic requests. Schedulability bounds are derived in order to perform off-line analysis and on-line reclaiming of any unused computation time.

1.1 Terminology and assumptions

We consider a hybrid set consisting of n firm periodic tasks and m soft aperiodic requests to be executed on a uniprocessor system. All tasks are preemptive and do not have precedence relations. Periodic tasks have deadlines, but they are allowed to occasionally skip one instance once in a while. Aperiodic requests do not have deadlines, and have to be executed as soon as possible to minimize their average response time. Periodic tasks are scheduled based on EDF, and aperiodic requests are handled by the Total Bandwidth Server (TBS) algorithm [15] (see section 4.1).

Each periodic task τ_i is characterized by a worst-case computation time c_i, a period p_i, a relative deadline equal to its period, and a skip parameter s_i, $2 \leq s_i \leq \infty$, which gives the minimum distance between two consecutive skips. When $s_i = \infty$ no skips are allowed and τ_i is equivalent to a hard periodic task. The skip parameter can be viewed as a *Quality Of Service* metric (the higher s, the better the quality of service).

Using the terminology introduced by Koren and Shasha in [6], every instance of a periodic task can be *red* or *blue*. A red instance must complete before its deadline; a blue instance can be aborted at any time. When a blue instance is aborted, we say that it was *skipped*. The fact that $s \geq 2$ implies that, if a blue instance is skipped, then the next $s - 1$ instances must be red. On the other hand, if a blue instance completes successfully, the next task instance is also blue.

2 Known results

In the basic periodic model in which all task instances are red (i.e., no skips are permitted), the schedulability of a periodic task set can be tested using a simple necessary and sufficient condition based upon cumulative processor utilization. In particular, Liu and Layland [9] showed that a periodic task set is schedulable by EDF if and only if its cumulative processor utilization is no greater than 1. That is,

$$\sum_{i=1}^{n} \frac{c_i}{p_i} \leq 1. \tag{1}$$

Analyzing the feasibility of periodic tasks that allow skip is not equally easy. Koren and Shasha [6] proved that determining whether a set of periodic occasionally skippable tasks is schedulable is NP-hard. They also found that, given a set $\Gamma = \{T_i(p_i, c_i, s_i)\}$ of periodic tasks that allow skips, then

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

is a necessary condition for the feasibility of Γ, since it represents the utilization based on the computation that must take place.

To better clarify the concepts mentioned above consider the task set shown in Table 1 and the corresponding feasible schedule, obtained by EDF, illustrated in Figure 1. Notice that the processor utilization factor is greater than 1 ($U_p = 1.25$), but condition (2) is satisfied.

Task	**Task1**	**Task2**	**Task3**
Computation	1	2	5
Period	3	4	12
Skip Parameter	4	3	*no skip*
U_p	1.25		

Table 1: A schedulable set of skippable periodic tasks.

Using a *processor demand* criterion, Jeffay and Stone [5] showed that a set of hard periodic tasks is schedulable by EDF if and only if, for any interval $L \geq 0$,

$$L \geq \sum_{i=1}^{n} \left\lfloor \frac{L}{p_i} \right\rfloor c_i. \tag{3}$$

Based on this result, Koren and Shasha proved the following theorem, which provides a sufficient condition for guaranteeing a set of skippable periodic tasks under EDF.

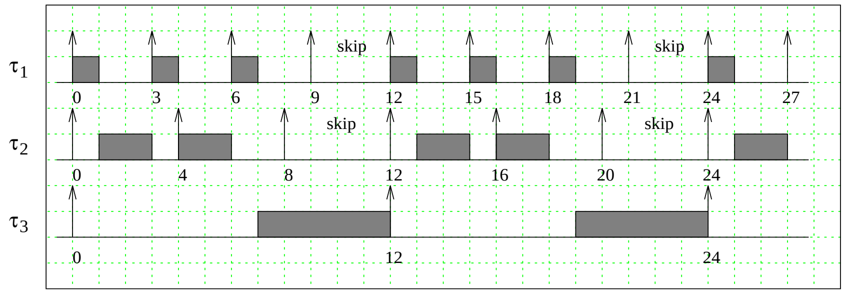

Figure 1: A feasible schedule of the task set shown in Table 1.

Theorem 1 *A set of firm (i.e., skippable) periodic tasks is schedulable if*

$$\forall L \geq 0 \qquad L \geq \sum_{i=1}^{n} D(i, [0, L]) \tag{4}$$

where

$$D(i, [0, L]) = \left(\left\lfloor \frac{L}{p_i} \right\rfloor - \left\lfloor \frac{L}{p_i s_i} \right\rfloor \right) c_i. \tag{5}$$

2.1 Algorithms

Two on-line scheduling algorithms were proposed in [6] to handle tasks with skips under EDF.

1. The first algorithm, called *Red Tasks Only* (RTO), always rejects the blue instances, whereas the red ones are scheduled according to EDF.

2. The second algorithm, called *Blue When Possible* (BWP), is more flexible than RTO and schedules blue instances whenever there are no ready red jobs to execute. Red instances are scheduled according to EDF.

It is easy to find examples that show that BWP improves RTO in the sense that it is able to schedule task sets that RTO cannot schedule. In the general case, the above algorithms are not optimal, but they become optimal under a particular task model, called the *deeply-red* model.

Definition 1 *A system is deeply-red if all tasks are synchronously activated and the first $s_i - 1$ instances of every task τ_i are red.*

In the same paper, Koren and Shasha showed that the worst case for a periodic skippable task set occurs when tasks are deeply-red. For this reason, all results in this paper will be proved assuming the assumption above. This means that, if a task set is schedulable under the deeply-red model, it is also schedulable without this assumption.

In the following sections, we address the problem of scheduling hybrid task sets, consisting of skippable periodic tasks and soft aperiodic requests. In particular, the RTO and the BWP algorithms will be integrated with the Total Bandwidth Server [13, 15] for enhancing aperiodic responsiveness. Before presenting the proposed approach, we introduce the notion of *equivalent utilization factor*, which simplifies the description of the schedulability analysis of hybrid task sets.

3 Equivalent utilization factor

In the basic periodic model, in which all tasks instances are red ($s_i = \infty$ for all tasks), the processor utilization factor is defined as $U_p = \sum_i c_i/p_i$. If skips are permitted in the periodic task set, the spare time saved by rejecting the blue instances can be reallocated for other purposes, e.g., for advancing the execution of soft aperiodic requests.

Unfortunately, the spare time has a "granular" distribution and cannot be reclaimed at any time. Nevertheless, we will show that skipping blue instances still produces a bandwidth saving in the periodic scheduling. To express this fact, we define an *equivalent utilization factor* U_p^* as follows.

Definition 2 *Given a set* $\Gamma = \{T_i(p_i, c_i, s_i)\}$ *of* n *periodic tasks that allow skips, we define the* equivalent processor utilization factor *as:*

$$U_p^* = \max_{L \geq 0} \left\{ \frac{\sum_i D(i, [0, L])}{L} \right\} \tag{6}$$

where

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

It is worth to observe that to compute U_p^* it is not necessary to evaluate equation (6) for any L, but it is sufficient to evaluate it in those points L that are periods' end points not beyond the hyperperiod $P = lcm(p_1, p_2, \ldots, p_n)$. The following theorem states the schedulability condition for a set of deeply-red skippable tasks.

Theorem 2 *A set* Γ *of skippable periodic tasks, which are deeply-red, is schedulable if and only if*

$$U_p^* \leq 1.$$

Proof.

If. Directly derives from Theorem 1 and the definition of U_p^*.

Only if. Suppose $U_p^* > 1$. If we consider any interval $[0, t]$, the total computation demand of Γ in the worst case is

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

and, according to the U_p^* definition,

$$\exists L \mid \quad U_p^* L = \sum_{i=1}^{n} D(i, [0, L]).$$

Hence

$$C[0, L] \;=\; \sum_{i=1}^{n} D(i, [0, L]) \;=\; U_p^* L \;>\; L.$$

Since the computation demand in $[0, L]$ exceeds the length of the same interval, Γ is not feasible. $\Box$

Notice that the factor U_p^* represents the net bandwidth really used by periodic tasks, assuming a deeply-red model. It is easy to show that

$$U_p^* \;\leq\; U_p.$$

In fact, by equation 3, U_p can also be defined as

$$U_p = \max_{L \geq 0} \left\{ \frac{\sum_i \left\lfloor \frac{L}{p_i} \right\rfloor c_i}{L} \right\}$$

Thus, $U_p^* \leq U_p$ because

$$\left(\left\lfloor \frac{L}{p_i} \right\rfloor - \left\lfloor \frac{L}{p_i s_i} \right\rfloor \right) \;\leq\; \left\lfloor \frac{L}{p_i} \right\rfloor.$$

4 Scheduling aperiodic tasks

In the previous section we showed that periodic skips create additional free bandwidth that can be used to advance the execution of soft aperiodic tasks. Such a free bandwidth can be exploited by a Total Bandwidth Server (TBS), a scheduling algorithm proposed by Spuri and Buttazzo [13, 15] to enhance aperiodic responsiveness under EDF. It is briefly recalled in the following section.

4.1 The Total Bandwidth Server

The name of the server comes from the fact that, each time an aperiodic request enters the system, the total bandwidth of the server (in terms of cpu execution time), is assigned to it, whenever possible. This is done by simply assigning a suitable deadline to the request, which is scheduled according to the EDF algorithm together with all the periodic task instances. The assignment of the deadline is done to improve the aperiodic responsiveness, still maintaining the schedulability of periodic tasks.

The definition of the TBS is relatively simple. When the k-th aperiodic request arrives at time $t = r_k$, it receives a deadline

$$d_k = \max(r_k, d_{k-1}) + \frac{C_k^a}{U_s}, \tag{7}$$

where C_k^a is the execution time of the request and U_s is the server utilization factor (i.e., its bandwidth). By definition $d_0 = 0$. The request is then inserted into the ready queue of the system and scheduled by EDF, as any periodic or sporadic instance.

Note that we can keep track of the bandwidth already assigned to other requests by simply taking the maximum between r_k and d_{k-1}. Intuitively, the assignment of the deadlines is such that in each interval of time the ratio allocated by EDF to aperiodic requests never exceeds the server utilization U_s, that is, the processor utilization of the aperiodic tasks is at most U_s. As a consequence, the schedulability of a periodic task set in the presence of a TBS can simply be tested by verifying the following condition:

$$U_p + U_s \leq 1 \tag{8}$$

where U_p is the utilization factor of the periodic task set.

In spite of its simplicity, the TBS shows a good performance/cost ratio and can easily be extended to deal with firm aperiodic requests, as described in [14]. We now show how to use the TBS to run in a real-time periodic environment which allow skips.

4.2 Integrating TBS with periodic skips

In this section we show how the spare time saved by skipped instances can be exploited by the TBS algorithm to advance the execution of aperiodic tasks, thus enhancing responsiveness with respect to the case of no skips. The following theorem gives a sufficient condition for guaranteeing a feasible schedule of a hybrid task set.

Theorem 3 *Given a set of periodic tasks that allow skip with equivalent utilization U_p^* and a set of soft aperiodic tasks served by a TBS with utilization factor U_s, the hybrid set is schedulable by RTO or BWP if:*

$$U_p^* + U_s \leq 1. \tag{9}$$

Proof.
See Appendix. □

Theorem 3 only provides a sufficient schedulability condition, since it guarantees a minimum bandwidth $U_{s_{min}} = 1 - U_p^*$ to the aperiodic tasks. Indeed, it is easy to find examples in which periodic task sets are schedulable by assigning the TBS a bandwidth U_s greater than $U_{s_{min}}$. The following theorem gives a maximum bandwidth $U_{s_{max}}$ above which the schedule is certainly not feasible.

Theorem 4 *Given a set $\Gamma = \{T_i(p_i, c_i, s_i)\}$ of n periodic tasks that allow skips, and a TBS with bandwidth U_s, then a necessary condition for the feasibility of Γ is:*

$$U_s \leq U_{s_{max}}$$

where

$$U_{s_{max}} = 1 - U_p + \sum_{i=1}^{n} \frac{c_i}{p_i s_i}. \tag{10}$$

Proof.
See Appendix. □

Notice that $U_{s_{max}}$ represents the limit case in which the whole bandwidth saved by skips can be used for aperiodic service. This is not always the case, since only a bandwidth $U_{s_{min}}$ can be guaranteed.

Table 2 shows a set of skippable tasks that can be feasibly scheduled with a TBS having a bandwidth $U_s > U_{s_{min}}$. Figure 2 and Figure 3 show the schedule produced by RTO and BWP respectively.

Task	**Task1**	**Task2**
Computation	2	2
Period	3	5
Skip Parameter	2	2
U_p	1.07	
U_p^*	0.8	
$U_{s_{min}}$	0.2	
$U_{s_{max}}$	0.47	

Table 2: A schedulable task set.

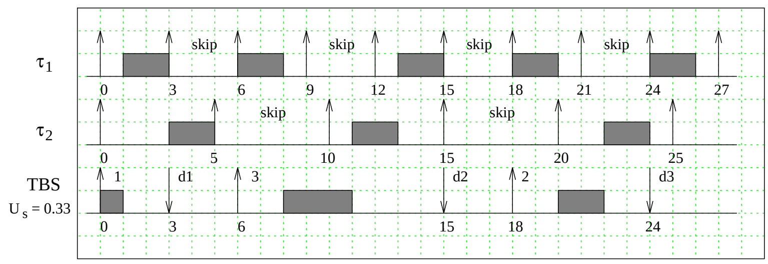

Figure 2: Schedule produced by RTO for the task set shown in Table 2.

The following theorem gives a condition in which $U_{s_{min}} = U_{s_{max}}$.

Theorem 5 *Given a feasible set* Γ *of* n *periodic tasks that allow skips, if*

$$U_p^* = \frac{\sum_{i=1}^{n} D(i, [0, H])}{H} \tag{11}$$

where

$$H = lcm(p_1 s_1, \dots, p_r s_r, p_{r+1}, \dots, p_n)$$

$$T_1, \dots, T_r \qquad task\ with \qquad S < \infty$$

$$T_{r+1}, \dots, T_n \qquad task\ with \qquad S = \infty$$

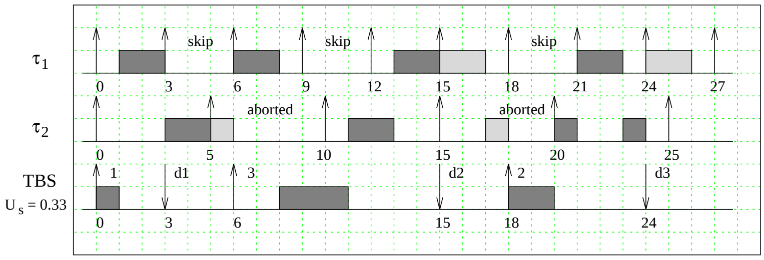

Figure 3: Schedule produced by BWP for the task set shown in Table 2.

then

$$U_{s_{min}} = U_{s_{max}}.$$

Proof.
See Appendix. □

The result of Theorem 5 can be used to adjust the parameters of the tasks as a function of the application requirements.

Notice that a small difference between $U_{s_{min}}$ and $U_{s_{max}}$ indicates that the spare time saved by skips tends to be uniformly distributed. When $U_{s_{min}} = U_{s_{max}}$, all skips are effectively used for aperiodic service, for any arrival pattern.

A big difference between $U_{s_{min}}$ and $U_{s_{max}}$ indicates that the spare time save by skips is not uniformly distributed, and only a bandwidth equal to $U_{s_{min}}$ can be guaranteed. In this case, the residual bandwidth $U_{s_{max}} - U_{s_{min}}$ cannot be assigned to the TBS but it can be exploited by the BWP to execute blue instances whenever possible.

An example of schedule produced by RTO and TBS on a task set in which $U_{s_{min}} = U_{s_{max}}$ is illustrated in Figure 4. Tasks parameters for this set are shown in Table 3.

Task	**Task1**	**Task2**	**Task3**
Computation	1	2	4
Period	3	5	15
Skip Parameter	5	3	*no skip*
U_p	1		
U_p^*	0.8		
U_s	0.2		
$U_{s_{max}}$	0.2		

Table 3: A schedulable tasks set

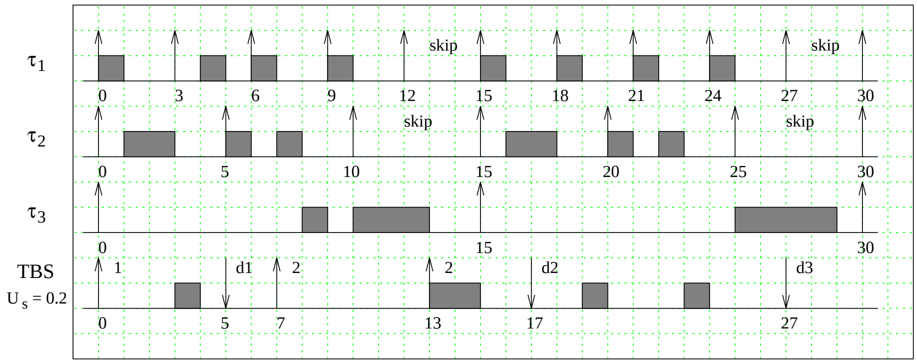

Figure 4: Schedule produced by RTO for the task set shown in Table 3.

5 Skip exploitation under TBS

In the previous section we noticed that when $U_{s_{min}} < U_{s_{max}}$ the spare time saved by skips is not uniformly distributed, hence there are intervals of time in which aperiodic tasks may execute without consuming the bandwidth reserved by the TBS. In this section, we show how to preserve the TBS bandwidth in the case in which aperiodic requests execute in such intervals. The proposed technique automatically reclaims the spare time saved by tasks (periodic or aperiodic) that may complete early. This is an important feature of this method, since the behavior of the TBS strictly depends on the estimated maximum execution time of periodic and aperiodic tasks, and, in the case of overestimated values, the deadlines assigned to aperiodic requests would be longer than necessary, thus aperiodic execution would be delayed.

The key idea for preserving the TBS bandwidth is to identify, after the execution of each aperiodic job, the earliest time t^* after which the aperiodic bandwidth U_s is totally available. This time is then used to compute the deadline for the next aperiodic request.

In order to compute t^* and guaranteeing the feasibility of the schedule with the new deadline assignment, some additional terminology has to be introduced.

5.1 Terminology and assumptions

In the definition of our reclaiming mechanism we will use the following notation:

- t: the instant at which the reclaiming algorithm is executed.
- $\tau_{i,j}$: denotes the j-th instance of periodic task τ_i.
- $\sigma_{i,j}$: denotes the current state of instance $\tau_{i,j}$; it is, a counter which identifies the number of consecutive red instances activated after a skipped blue instance; $\sigma_{i,j} \in [0, s_i - 1]$; $\sigma_{i,j} = 0$ when the current job is blue.
- $c_{i,j}$: denotes the worst case execution time of instance $\tau_{i,j}$.
- $r_{i,j}$: denotes the arrival time of instance $\tau_{i,j}$, i.e., the time at which the job is activated and becomes ready to execute.

- $c_i^r(t)$: denotes the remaining computation time of the current instance of periodic task τ_i, i.e., the residual worst case execution time needed by the processor, at the current time t, to complete the instance without interruption; by definition, $c_i^r(t) = 0$ either if the current instance is blue, or $t = r_{i,j}$, for a $j > 0$.

- $D(i, [t_1, t_2])$: denotes the computation demand of red instances of periodic task τ_i during the interval $[t_1, t_2]$; that is, the processing time of those red instances of τ_i released at or after t_1 and with deadline less than or equal to t_2. More formally,

$$D(i, [t_1, t_2]) = \sum_{r_{i,j} \geq t_1, d_{i,j} \leq t_2, \sigma_{i,j} \neq 0} c_{i,j}.$$

- $A(i, [t_1, t_2])$: denotes the computation time requested by τ_i in the interval $[t_1, t_2)$, that is, the remaining computation time of the current instance of periodic task τ_i plus the computation time of the red jobs released in $[t_1, t_2)$. More formally,

$$A(i, [t_1, t_2]) = \begin{cases} c_i^r(t_1) + \sum_{t_1 \leq r_{i,j} < t_2}^{\sigma_{i,j} \neq 0} c_{i,j} & \text{if } t_2 > t_1 \\ c_i^r(t_1) & \text{if } t_2 = t_1 \end{cases}$$

- $D_T(t_1, t_2)$: denotes the total computation demand of red instances of all periodic tasks during the interval $[t_1, t_2]$; that is, the processing time of those red instances released at or after t_1 and with deadline less than or equal to t_2. More formally,

$$D_T(t_1, t_2) = \sum_{d_i \leq t_2} c_i^r(t_1) + \sum_i D(i, [t_1, t_2]).$$

For a better exploitation of skips, throughout our analysis we assume that periodic tasks are scheduled by the RTO algorithm.

5.2 The reclaiming algorithm

Whenever an aperiodic request J_{k-1} is completed, the next pending request J_k, if any, becomes *eligible* to execute. Let t be such a time. Based on the classical TBS, at time t, J_k receives a deadline $d_k = max(t, d_{k-1}) + \frac{C_k^a}{U_s}$ and it is scheduled by EDF along with the periodic task instances. Such a deadline assignment guarantees that the periodic task set remains schedulable [15].

As we observed above, however, if job J_{k-1} does not entirely consume the available bandwidth U_s (either because it exploited skips or because it executed less than its worst case duration) we can exploit the unused bandwidth by advancing deadline d_k of job J_k. To advance d_k, our reclaiming algorithm starts finding the earliest time t^* ($t \leq t^* \leq d_{k-1}$) after which the total bandwidth U_s is again available for aperiodic service.

Time t^* can be computed by observing that for all $L > t^*$ the total computation demand of periodic and aperiodic tasks must be less than or equal to the available processing time. In fact, the next theorem follows:

Theorem 6 *Given a set Γ of periodic tasks that allow skips and a TBS such that $U_p^* + U_s \leq 1$, if we have the early end of an aperiodic task in t, we can compute the time t^* as follows:*

$$t^* = \min_{x \geq t}\{x \mid \forall L \geq x \;\; D_T(t, L) + (L - x)U_s \leq L - t)\}. \tag{12}$$

Proof.
See Appendix. □

Clearly, if $t \geq d_{k-1}$, no reclaiming can be done (since the bandwidth U_s is already guaranteed by the TBS) and we can immediately set $t^* = t$.

Since, after t^*, the total bandwidth U_s is available, the eligible aperiodic task J_k will be assigned a deadline d_k according to the following TBS rule:

$$d_k = t^* + \frac{C_k^a}{U_s}. \tag{13}$$

The algorithm for computing t^* is shown in Figure 5. It uses the following function `next_time(t)` which returns the next request $r_{i,j} \geq t$ among all periodic tasks:

$$next\_time(t) = \min_{i,j}\{r_{i,j} \mid \; r_{i,j} \geq t\}.$$

Notice that t^* can be computed only using L values which are periods' end points of periodic tasks, up to a maximum time t_{max}.

```
reclaiming(t_max, d_{k-1})      /* returns t* */
{
        t = current_time();
        L = next_time(t);
        t* = t;

        do {
                newt  =  L - [L - t - D_T(t, L)]/U_s;
                t* = max(newt, t*);

                if (t* >= d_{k-1}) return(d_{k-1});

                L = next_time(L + 1);

        } while (L <= t_max);
        return(t*);
}
```

Figure 5: Algorithm of t^* computation.

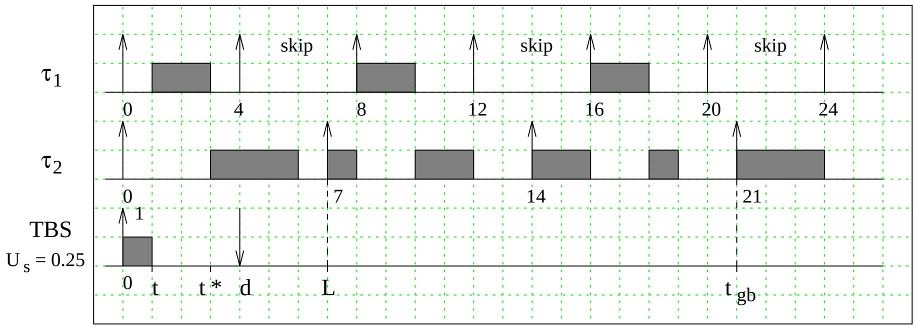

Figure 6: Example of t^* computation.

Figure 6 illustrates an example of t^* computation performed at time $t = 1$. We can note that $L = 7$ is the instant which determines the value of t^*, that is $t^* = 3$.

We now restrict the value of t_{max} by introducing the notion of generalized busy-period, called *GB-period*, which gives the interval of continuous processor utilization due to the red periodic instances and the bandwidth U_s. More formally, at any time t, the GB-period(t) is that interval $[t, t_{gb}]$ such that:

$$t_{gb} = \min_{x \geq t} \left\{ x \,\middle|\, \sum_i A(i, [t, x]) + (x - t)U_s \leq x - t \right\}. \tag{14}$$

The following theorem guarantees that the number of points L, in the t^* computation, can be limited in the range $[t, t_{gb}]$.

Theorem 7 *Let Γ be a hybrid set of tasks such that $U_p^* + U_s \leq 1$; let t be the instant at which the aperiodic task J_k becomes eligible and a reclaiming occurs. Then, if no task misses its deadline in $[t, t_{gb}]$, then Γ is schedulable.*

Proof.
See Appendix. $\square$

The algorithm for computing the GB-period is reported in Figure 7. Function `next(t)` is used to compute the next request $r_{i,j} \geq t$ among all periodic <u>red</u> instances:

$$next(t) = \min_{i,j} \{ r_{i,j} \mid \; (\sigma_{i,j} > 0) \wedge (r_{i,j} \geq t) \} .$$

Function `req(t)` is used to compute the time requested by periodic <u>red</u> instances with $r_{i,j} = t$:

$$req(t) = \sum_{r_{i,j} = t, \sigma_{i,j} > 0} c_{i,j}.$$

Note that this algorithm returns $next(t_{gb})$. An example of GB-period(t) computation is shown in Figure 8. Here, points L_1, L_2, L_3 indicate the times at which $l = last\_l$ and variable BAR represents the bandwidth reserved to aperiodic requests. The termination condition of the cycle is computed for each of these three points, but it is satisfied only for L_3, obtaining $t_{gb} = 21$.

```
gb_period(t)            /* returns t_gb */
{
    BAR = 0;
    l = t;

    while (TRUE) {
        last_l = l;
        l = t + Σ_i A(i, [t, l]) + BAR;

        if (l == last_l) {
            new = next(l);
            if (Σ_i A(i, [t, new])+
                (new − t) ∗ U_s ≤ new − t)
                    return(new);
            else {
                BAR = BAR + new − l;
                l = new + req(new);
            }
        }
    }
}
```

Figure 7: Algorithm for GB-period(t).

6 Experimental results

In this section we present some results of the simulations we have carried out for evaluating the performance of our reclaiming algorithm (TBrec) with respect to the plain TBS and the background service, for different aperiodic loads and skip parameters. The execution times of aperiodic requests are choosen to be uniformly distributed in the interval $[5, 20]$, whereas their interarrival times are generated according to an exponential distribution, with average value computed to impose a specific aperiodic load. In the graphs, the average response time of aperiodic tasks is plotted as a function of the aperiodic load and is normalized with respect to their computation time. Hence, a value of 5 on the y-axis means that the average response time of aperiodic tasks is five times longer than their average computation time.

The experiments refer to a task set of five periodic tasks with $U_p = 0.8375$, however the skip parameters used in four simulations of Figure 9, Figure 10, Figure 11 and Figure 12 are different; in Table 4 we can see the task set parameters used to generate periodic load. We can note that the skip parameters are increased in every simulation (less skips are allowed), in fact the skip parameters used in Figure 9 are less than those used in Figure 10 and so on.

As shown in Figure 9, for small skip parameters (i.e., when many instances are skipped) the performance of TBS tends to be similar to that of background service. Nevertheless, TBrec is able to achieve good performance by exploiting the execution time left by the skipped (blue)

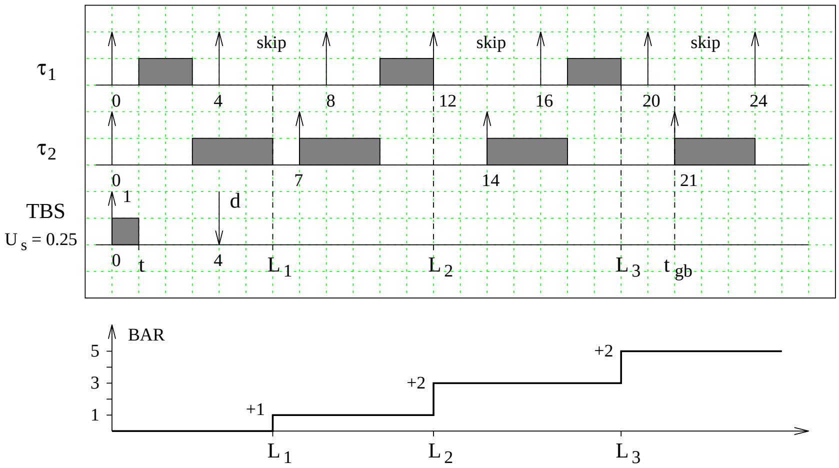

Figure 8: Example of t_{gb} computation.

Task	**Task1**	**Task2**	**Task3**	**Task4**	**Task5**
Computation	8	20	10	20	30
Period	60	80	100	120	160
Sim.1: Skip	5	4	4	5	∞
Sim.2: Skip	5	5	5	∞	∞
Sim.3: Skip	5	6	8	∞	∞
Sim.4: Skip	∞	∞	∞	∞	∞

Table 4: Simulations parameters

instances. As shown in Figure 10 and Figure 11, the improvement achieved by TBrec is even more significant for high aperiodic loads and high skip parameters (i.e., when less instances are skipped). In this situation, in fact, the background exhibits poor performance (since less “holes” are available), whereas the TBS degrades for high load conditions. When no skips are permitted (see Figure 12), the periodic task set becomes hard and background service exhibits the worst performance. Also in this case TBrec improves the TBS algorithm for high aperiodic loads because it reclaims the band unused by the server when there are no pending aperiodic requests.

In summary, experimental results indicate that the our reclaiming algorithm is worth to be used when a high quality of service is required by periodic tasks and for high load conditions. When load is not so high the plain TBS is able to achieve acceptable performance, thus the reclaiming mechanism might be avoided to limit the runtime overhead.

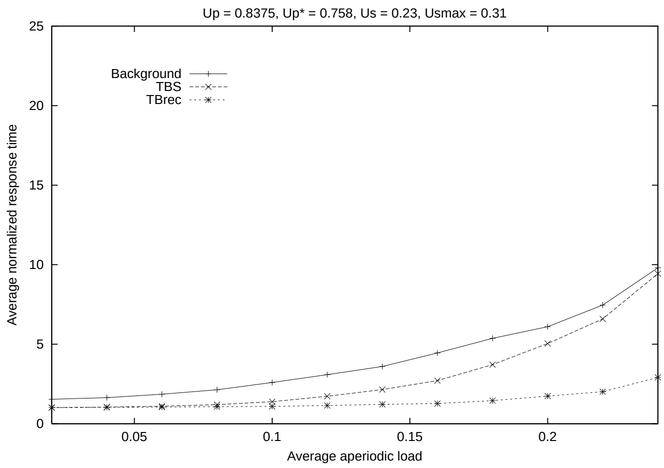

Figure 9: Performance results with simulation 1.

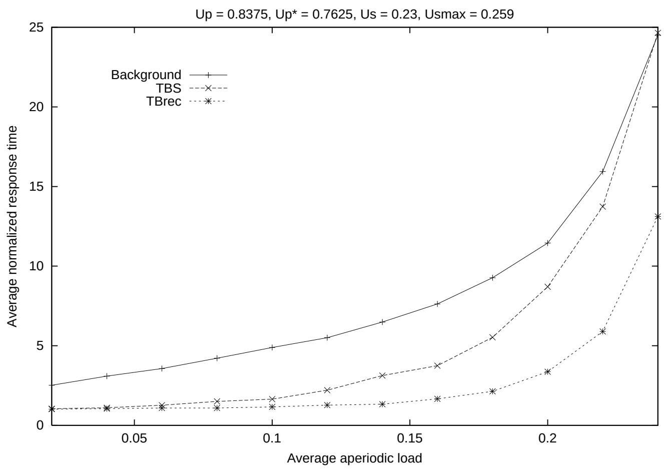

Figure 10: Performance results with simulation 2.

7 Conclusions

In this paper, we addressed the problem of scheduling hybrid task sets consisting of periodic tasks that can occasionally skip one instance and soft aperiodic requests, which have to be served as soon as possible to minimize their average response time. We proposed and analyzed an algorithm, based on a variant of Earliest Deadline First scheduling, which exploits skips to enhance aperiodic responsiveness. Schedulability bounds have been derived to ensure a minimum level of guarantee off-line. In particular, we showed that the spare time saved by skipping periodic instances creates a free bandwidth that can be used either for scheduling slightly overloaded systems or for reducing the average response time of aperiodic requests.

Since the spare time saved by skips is not uniformly distributed, we proposed a reclaiming algorithm for preserving the unused bandwidth when aperiodic requests execute in intervals

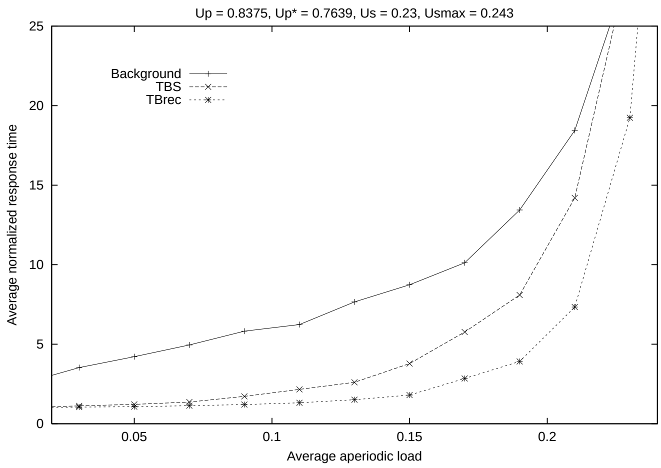

Figure 11: Performance results with simulation 3.

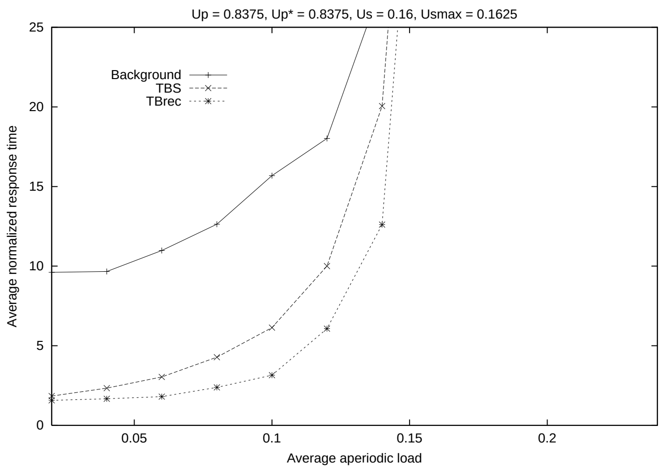

Figure 12: Performance results with simulation 4.

not explicitly reserved to the TBS. The proposed technique automatically reclaims the spare time saved by tasks that may complete early. Experimental results indicate that reclaiming is more effective for high loads and when high quality of service is required. We are currently investigating possible extensions of this approach to handle tasks with deadline and fault-tolerant requirements.

Appendix

Proof of Theorem 3.

Assume $U_p^* + U_s \leq 1$, and suppose that a time-overflow occurs at time t. Let $t_a \geq 0$ be the last time before t at which the CPU is not running red tasks; let $t_b \geq 0$ be the last time before t at which the CPU is running red tasks with deadlines after t. If we take $t' = \max\{t_a, t_b\}$ (see

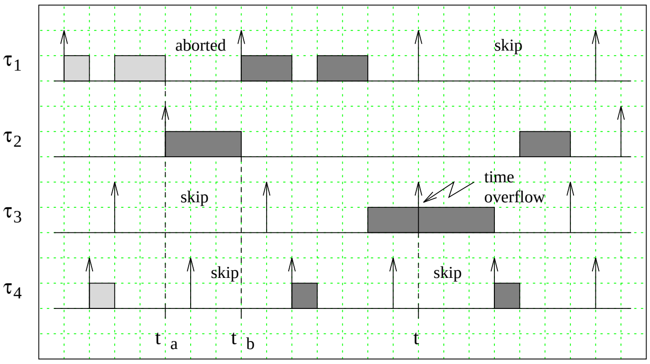

Figure 13: Example for the proof of Theorem 3.

Figure 13), time t' has the property that only red tasks activated after t' with deadlines less than or equal to t run during $[t', t]$. We can notice that:

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

Let C be the total computation demand requested in the interval $[t', t]$. Since a time-overflow occurs, it must be that:

$$t - t' < C.$$

Let C_{ape} be the total execution time actually demanded by aperiodic requests arrived at t' or later and served with deadlines less than or equal to t:

$$C_{ape} \;=\; \sum_{r_k \geq t', d_k \leq t} C_k^a.$$

By the result of Lemma 2 proved in [15], which says that $C_{ape} \leq (t - t')U_s$, we can write that

$$\begin{aligned} C \;&=\; \sum_i D(i, [t', t]) \;+\; C_{ape} \;\leq \\ &\leq\; \sum_i D(i, [t', t]) \;+\; U_s(t - t') \;\leq \\ &\leq\; \sum_i D(i, [0, t - t']) \;+\; U_s(t - t') \;\leq \\ &\leq\; U_p^*(t - t') \;+\; U_s(t - t'). \end{aligned}$$

As a consequence,

$$t - t' \;<\; C \;\leq\; U_p^*(t - t') + U_s(t - t')$$

and hence

$$U_p^* + U_s \;>\; 1$$

which is a contradiction. □

Proof of Theorem 4

Given a periodic tasks set Γ, suppose to schedule the set $\Gamma' = \Gamma \cup \{T(p,c,s)\}$, such that:

$$\forall p, \qquad c > pU_{s_{max}}, \qquad s = \infty$$

$$U_{s_{max}} = 1 - \sum_{i=1}^{n} \left(\frac{c_i}{p_i} - \frac{c_i}{p_i s_i} \right)$$

Now, checking the necessary condition 2 for Γ' we have:

$$\sum_{i=1}^{n+1} \left(\frac{c_i}{p_i} - \frac{c_i}{p_i s_i} \right) = \sum_{i=1}^{n} \left(\frac{c_i}{p_i} - \frac{c_i}{p_i s_i} \right) + \frac{c}{p} >$$

$$> \sum_{i=1}^{n} \left(\frac{c_i}{p_i} - \frac{c_i}{p_i s_i} \right) + U_{s_{max}} = 1.$$

Since Γ' does not satisfy condition (2), Γ' is not feasible. $\square$

Proof of Theorem 5

$$U_{s_{min}} - U_{s_{max}} = 1 - U_p^* - 1 + U_p - \sum_{i=1}^{r} \frac{c_i}{p_i s_i} =$$

$$= U_p - \frac{\sum_i D(i,[0,H])}{H} - \sum_{i=1}^{r} \frac{c_i}{p_i s_i} =$$

$$= U_p - \frac{\sum_{i=1}^{n} \left\lfloor \frac{H}{p_i} \right\rfloor c_i - \sum_{i=1}^{r} \left\lfloor \frac{H}{p_i s_i} \right\rfloor c_i}{H} - \sum_{i=1}^{r} \frac{c_i}{p_i s_i} =$$

$$= U_p - \sum_{i=1}^{n} \frac{c_i}{p_i} + \sum_{i=1}^{r} \frac{c_i}{p_i s_i} - \sum_{i=1}^{r} \frac{c_i}{p_i s_i} = 0.$$

Thus, the theorem follows. $\square$

Lemma 1 *Given a set Γ of periodic tasks that allow skips and a TBS such that $U_p^* + U_s \leq 1$, if we have the early end of an aperiodic task in t and a resource-reclaiming is executed modifing the scheduling after t, supposing that a time-overflow occurs in $t_2 > t$; then it must happen that $t_1 \leq t$, defined t_1 as the last instant before t_2 in which the cpu doesn't execute red tasks with deadline less than or equal to t_2.*

Proof.

Suppose that a time-overflow occurs at time t_2 and $t_1 > t$; an active task τ_k with request $r_k < t_1$ and deadline $d_k \leq t_2$ can't exist in t_1, besides time t_1 has the property that only red jobs activated after t_1 with deadlines less than or equal to t_2 run during $[t_1, t_2]$. Let C be the

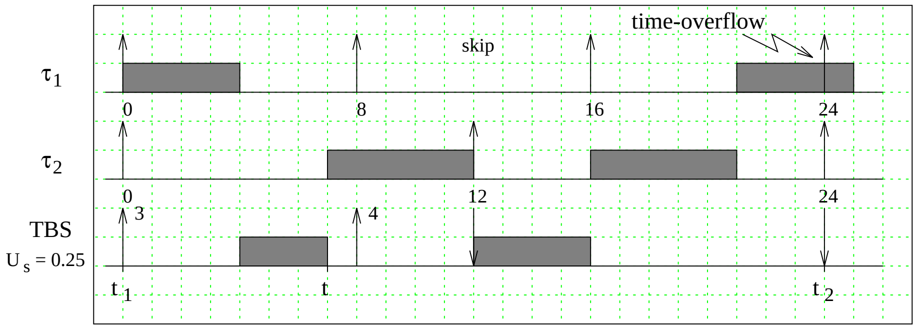

Figure 14: Example for the Lemma 1.

total computation demand requested in the interval $[t_1, t_2]$, since a time-overflow occurs in t_2, then:

$$C > t_2 - t_1$$

Let C_{ape} be the total execution time actually demanded by aperiodic requests arrived at t_1 or later and served with deadlines less than or equal to t_2:

$$C_{ape} = \sum_{r_k \geq t_1, d_k \leq t_2} C_k^a \leq (t_2 - t_1)U_s$$

$$C = \sum_i D(i, [t_1, t_2]) + C_{ape} \leq U_p^*(t_2 - t_1) + U_s(t_2 - t_1)$$

Hence,

$$(U_s + U_p^*)(t_2 - t_1) > t_2 - t_1$$

$$\Rightarrow \quad U_s + U_p^* > 1$$

Which is a contradiction. □

Proof of Theorem 6

Supposing that a time-overflow occurs in $t_{ov} > t^*$ and that the condition (12) is verified; by the lemma 1 we can write that:

$$\sum_{d_i(t) \leq t_{ov}} c_i^r(t) + \sum_i D(i, [t, t_{ov}]) + C_{ape} > t_{ov} - t$$

With C_{ape} the total execution time actually demanded by aperiodic requests arrived at t^* or later and served with deadlines less than or equal to t_{ov}:

$$C_{ape} \leq (t_{ov} - t^*)U_s$$

$$\sum_{d_i(t) \leq t_{ov}} c_i^r(t) + \sum_i D(i, [t, t_{ov}]) + U_s(t_{ov} - t^*) > t_{ov} - t$$

Which is a contradiction. □

Proof of Theorem 7

Suppose that a time-overflow occurs in $t_{ov} > t_{gb}$; we can write that:

$$\sum_{d_i \leq t_{ov}} c_i^r(t) \; + \; \sum_i D(i, [t, t_{ov}]) \; + \; C_{ape} \; > \; t_{ov} - t \tag{15}$$

where C_{ape} is the total execution time actually demanded by aperiodic requests arrived at t or later and served with deadlines less than or equal to t_{ov}:

$$C_{ape} \;\; = \sum_{r_k \geq t, d_k \leq t_{ov}} C_k^a \;\; \leq \;\; U_s(t_{ov} - t)$$

For a generic $t \leq \tilde{t} < t_{ov}$, we can write that

$$\sum_{d_i \leq t_{ov}} c_i^r(t) \; + \; \sum_i D(i, [t, t_{ov}]) \;\; \leq \;\; \sum_i A(i, [t, \tilde{t}]) \; + \; \sum_i D(i, [\tilde{t}, t_{ov}]).$$

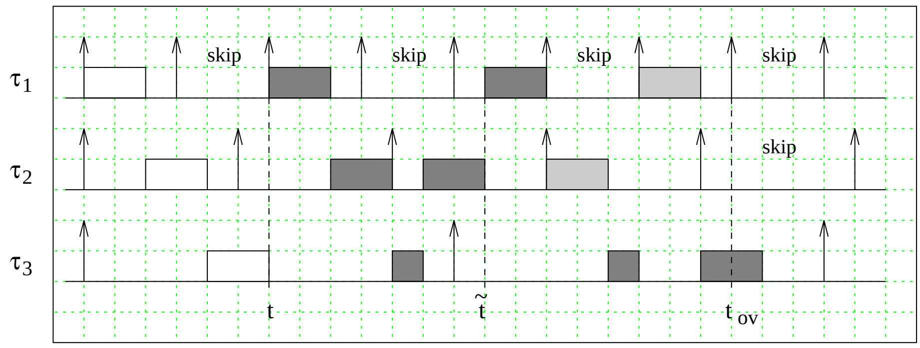

Figure 15: Example for the proof of Theorem 7.

By the definition of GB-period we can also write that

$$\sum_i A(i, [t, t_{gb}]) \; + \; U_s(t_{gb} - t) \; \leq \; t_{gb} - t.$$

Hence, from equation (15), setting $\tilde{t} = t_{gb}$, we obtain:

$$\sum_i A(i, [t, t_{gb}]) \; + \; \sum_i D(i, [t_{gb}, t_{ov}]) \; + \; U_s(t_{ov} - t) \;\; > \;\; t_{ov} - t$$

$$\sum_i A(i, [t, t_{gb}]) \; + \; \sum_i D(i, [t_{gb}, t_{ov}]) \; + \; U_s(t_{gb} - t + t_{ov} - t_{gb}) \;\; > \;\; t_{ov} - t$$

$$(t_{gb} - t) \; + \; \sum_i D(i, [t_{gb}, t_{ov}]) \; + \; U_s(t_{ov} - t_{gb}) \;\; > \;\; (t_{ov} - t_{gb}) \; + \; (t_{gb} - t)$$

$$\sum_i D(i,[t_{gb},t_{ov}]) \; + \; U_s(t_{ov}-t_{gb}) \;\; > \;\; (t_{ov}-t_{gb})$$

Finally, by the definition of U_p^*, we have:

$$\sum_i D(i,[t_{gb},t_{ov}]) \; \leq \; U_p^*(t_{ov}-t_{gb})$$

$$(U_s+U_p^*)(t_{ov}-t_{gb}) \; > \; (t_{ov}-t_{gb})$$

$$U_s+U_p^* \; > \; 1$$

which is a contradiction. $\Box$

References

[1] N.C. Audsley, A. Burns, M. Richardson, K. Tindell and A. Wellings, "Applying New Scheduling Theory to Static Priority Preemptive Scheduling", *Software Engineering Journal* 8(5), pp. 284-292, September 1993.

[2] T.P. Baker, "Stack-Based Scheduling of Real-Time Processes," *Real-Time Systems*, 3, 1991.

[3] T.M. Ghazalie and T.P. Baker, "Aperiodic Servers In A Deadline Scheduling Environment," *Real-Time Systems*, 9, pp. 21–36, 1995.

[4] M. Hamdaoui and P. Ramanathan, "A Dynamic Priority Assignment Technique for Streams with (m,k)-Firm Deadlines," *IEEE Transactions on Computers*, 44(12), December 1995.

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

[6] G. Koren and D. Shasha, "Skip-Over: Algorithms and Complexity for Overloaded Systems that Allow Skips," *Proceedings of IEEE Real-Time System Symposium*, Pisa, Italy, December 1995.

[7] J.P. Lehoczky, L. Sha and J.K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," *Proc. of the IEEE Real-Time Systems Symposium*, December 1987.

[8] 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 IEEE Real-Time Systems Symposium*, pp. 110-123, Phoenix, Arizona, December 1992.

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

[10] A. K. Mok and D. Chen, "A Multiframe Model for Real-Time Tasks", *Proceedings of IEEE Real-Time System Symposium*, Washington DC, December 1996.

[11] L. Sha, R. Rajkumar and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, 39(9), September 1990.

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

[13] Spuri, M. and G. C. Buttazzo, "Efficient Aperiodic Service under Earliest Deadline Scheduling", *Proceedings of IEEE Real-Time System Symposium*, San Juan, Portorico, December 1994.

[14] M. Spuri, G.C. Buttazzo, and F. Sensini, "Robust Aperiodic Scheduling under Dynamic Priority Systems", *Proc. of the IEEE Real-Time Systems Symposium*, Pisa, Italy, December 1995.

[15] M. Spuri and G.C. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems," *Real-Time Systems*, 10(2), 1996.