Technical Report

Uniprocessor EDF Scheduling with Mode Change

Björn Andersson

HURRAY-TR-081001

Version: 0

Date: 10-01-2008

Uniprocessor EDF Scheduling with Mode Change

Björn Andersson

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: bandersson@dei.isep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

Consider the problem of scheduling sporadically-arriving tasks with implicit deadlines using Earliest-Deadline-First (EDF) on a single processor. The system may undergo changes in its operational modes and therefore the characteristics of the task set may change at run-time. We consider a well-established previously published mode-change protocol by Sha et al. and we show that if every mode utilizes at most 50% of the processing capacity then all deadlines are met. We also show that there exists a task set that misses a deadline although the utilization exceeds 50% by just an arbitrarily small amount. Finally, we present, for a relevant special case, an exact schedulability test for EDF with mode change.

Uniprocessor EDF Scheduling with Mode Change

Björn Andersson

IPP Hurray Research Group,
Polytechnic Institute of Porto, Portugal

Abstract. Consider the problem of scheduling sporadically-arriving tasks with implicit deadlines using Earliest-Deadline-First (EDF) on a single processor. The system may undergo changes in its operational modes and therefore the characteristics of the task set may change at run-time. We consider a well-established previously published mode-change protocol by Sha et al. and we show that if every mode utilizes at most 50% of the processing capacity then all deadlines are met. We also show that there exists a task set that misses a deadline although the utilization exceeds 50% by just an arbitrarily small amount. Finally, we present, for a relevant special case, an exact schedulability test for EDF with mode change.

1 Introduction

Many real-time systems need to reconfigure themselves during operation and thereby change the characteristics of their tasks. There are four common reasons for the need to reconfigure. First, a distributed computer system may suffer from a permanent fault in one of its computer nodes and as a result it is necessary that the software system reconfigures itself (to use less resources) on the non-faulty computer nodes in order to run the software tasks that were residing on the computer node that suffered from a fault. For example, algorithms that perform feedback control of physical objects can often be implemented in different ways with some implementations offering high performance (for example low error in the measured variable) at the cost of a long execution time whereas other implementations offer less performance but with a smaller execution time. The second reason for the need to reconfigure is that the physical operating environment is changed. For example, the software that needs to execute when an aircraft is taking off is not the same as the software needed when the aircraft is flying at high altitude. The third reason for the need to configure is that the computer system offers service to many users/objects and the number of users/objects varies at run-time. For example, consider a computer system for tracking objects, such as aircrafts and helicopters [1]. When the number of objects increases, it may not be possible to perform tracking with the highest accuracy for all tasks and hence the software must reconfigure to use a lower accuracy for some of its tracking tasks. The fourth reason for the need to configure is that the software system must be updated during operation.

Reconfiguration requires that decisions are taken on (i) which configuration should be used and (ii) once a configuration has been selected, how to transition the software to this new configuration. The former is application-dependent and therefore resulted in a large number of studies (see for example [1–4]). The latter involves the problem of ensuring that all computer nodes agree on the new configuration and ensuring that software in the new configuration does not misinterpret data structures that were written to in the old configuration. These issues are application dependent as well. Another issue is that the software in the old configuration as well as the new configuration may be proven to meet deadlines if run in isolation but the reconfiguration may cause software tasks to arrive in a pattern which was not analyzed in each of the individual configurations and hence a deadline can be missed. It is therefore crucial to (i) design a protocol (called a *mode-change protocol*) that prescribes how tasks are allowed to arrive during the reconfiguration and (ii) design a method (a schedulability test) for proving that deadlines are met during the reconfiguration. This is often called the *mode-change problem*.

The research literature offers solutions to the mode-change problem. Traditionally, mode changes in table-driven time-triggered systems were performed by letting all tasks switch to a new mode in the beginning of the table. This approach is simple to understand, implement and it does not constrain the schedulability of the system. But unfortunately, it may take a long time from when a request is made to switch the task set to a new mode until all tasks have switched to the new mode. For event-triggered systems, it has been found for static-priority scheduling that switching to the new mode when a processor becomes idle does not jeopardize the timeliness guarantees proven for the respective configuration [5]. But it has the same drawback as the mode-change approach for table-driven time triggered systems; it may take a long time to perform the mode change. A mode-change protocol which allows a much faster mode change has been proposed [6] by Sha et al.; when a mode change request occurs, the currently released job of a task continues to execute according to its old configuration but the future released jobs will be given parameters (execution time and minimum inter-arrival time) by the new configuration. A correct schedulability analysis for this mode change protocol was later proposed as well [7] but only for static-priority scheduling. In fact, the research literature offers no mode-change protocol and corresponding schedulability analysis for a processor scheduled by the algorithm Earliest-Deadline-First (EDF) [8]. This is unfortunate because EDF is capable of scheduling workloads that static-priority scheduling is unable to schedule [8].

Therefore, in this paper, we present a new solution to the mode change problem. We use the previously known, and well established, mode-change protocol designed by Sha et al. [6]. But we use EDF and we present a schedulability analysis for EDF with mode changes. The schedulability analysis is simple; if the utilization of every mode is at most 50% then all deadlines are met. We also show that there exists a task set that misses a deadline although the utiliza-

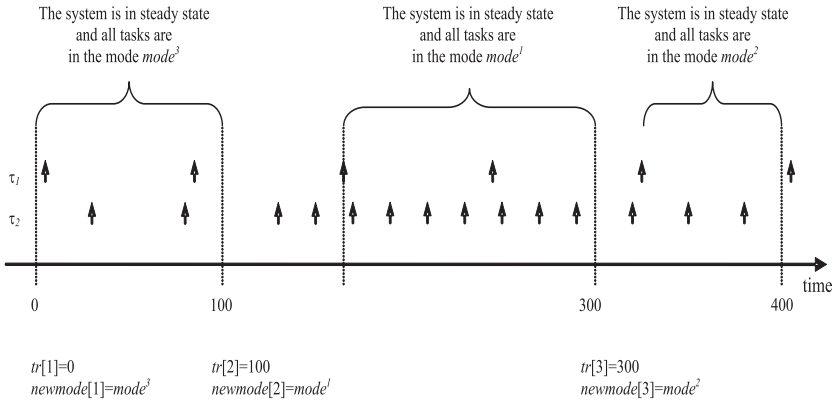

Fig. 1. An example of a task set $\tau = \{\tau_1, \tau_2\}$ and mode set $modes = \{mode^1, mode^2, mode^3\}$ and how the task set τ undergoes mode changes. During the time interval [0,100), all tasks are in $mode^3$. During the time interval [165,300), all tasks are in $mode^1$. During the time interval [325,400), all tasks are in $mode^2$. The vertical lines with arrows pointing upwards show the arrivals of tasks. In this particular example, tasks arrive as frequently as possible given by their specified minimum inter-arrival times. The minimum inter-arrival time of task τ_2 is different in different modes; in $mode^3$ it is 50, whereas in $mode^1$ it is 20 and in $mode^2$ it is 30. It can be seen that although a mode change is requested at time 100, task τ_2 has to wait until time 130 until it changes to $mode^1$ and τ_1 has to wait until time 165 until it switches to $mode^1$. It can also be seen that although τ_1 changes its modes, in this particular example, the minimum inter-arrival time of τ_1 remains the same in all modes.

tion exceeds 50% by just an arbitrarily small amount. Finally, we present, for a relevant special case, an exact schedulability test for EDF with mode change.

The remainder of this paper is organized as follows. Section 2 gives the system model. Section 3 presents the utilization bound (50%) and shows that the bound derived is tight. Section 4 presents a new schedulability test. Section 5 gives conclusions.

2 System model

Figure 1 illustrates concepts that we use. Consider a task set $\tau = \{\tau_1, \tau_2, \tau_3, \ldots\}$ and a mode set $modes = \{mode^1, mode^2, mode^3, \ldots\}$. Also, consider a sequence of times of transition requests $< tr[1], tr[2], tr[3], \ldots >$ and corresponding new modes $< newmode[1], newmode[2], newmode[3], \ldots >$ where each of those modes are in the set modes. These two sequences have the interpretation that the current mode of the task set τ is requested to become $newmode[j]$ at time $tr[j]$. We assume that the request of the transition of the task set to mode $newmode[j]$ at time $tr[j]$ is unknown to the scheduling algorithm and mode change protocol before time $tr[j]$.

A task τ_i generates a (potentially infinite) sequence of jobs. We consider the sporadic model, that is, the time of the arrival of a job is unknown before the job arrives and the arrival time of a job cannot be controlled by the scheduling algorithm. A task τ_i has a current mode at time t; this mode is one of the modes in the set modes. A task τ_i is characterized by the minimum inter-arrival time of task τ_i in mode k (denoted T_i^k) and the execution time of task τ_i in mode k (denoted C_i^k). The parameters T_i^k and C_i^k have the following interpretation. If task τ_i is in mode $mode^k$ at time t and s denotes the latest time not exceeding t when task τ_i has arrived then it holds that the next arrival of task τ_i occurs at time $s + T_i^k$ or later. If task τ_i is in mode $mode^k$ at time t and task τ_i has never arrived before time t then it is possible for τ_i to arrive at time t. Let us consider a job of task τ_i that arrives at time s and the job is in mode k at that time, time s. Then the deadline of the job is $s + D_i^k$. If the job performs C_i^k time units of execution by its deadline then we say that the job meets its deadline; otherwise it misses its deadline. A task τ_i is said to meet its deadlines if all of its jobs meet their deadlines; otherwise the task τ_i is said to miss a deadline. A task set τ is said to meet its deadlines if all tasks in τ meet their deadlines otherwise we say that the task set τ misses a deadline. We assume the implicit deadline model, that is, $\forall i, k : D_i^k = T_i^k$.

We assume that preemptive Earliest-Deadline-First (EDF) scheduling is used to schedule jobs on a single processor. It operates as follows. At time t, the processor selects for execution the job with the earliest deadline among the tasks for which both of the following conditions are true: (i) the arrival time of the job is no later than time t and (ii) the remaining execution time of the job at time t is strictly greater than zero. If two or more jobs have the same deadline then any of those jobs can be selected; the tie-breaking of priorities between jobs is arbitrarily and this tie-breaking does not need to be consistent throughout time.

We say that the system is in steady state at time t if it holds that all tasks are in the same mode at time t. We say that the system is in transient state at time t if it is not in steady state at time t. We let $latest\_arrival(t, \tau_i)$ denote the maximum time such that (i) this time is no greater than t and (ii) task τ_i arrives at time t. If task τ_i has not yet arrived at time t, then $latest\_arrival(t, \tau_i)$ is undefined. Let us assume that the system has a variable $pending\_mode\_changes$, a set which is initialized to the empty set when the system boots.

If the system is in steady state at time t and all tasks are in mode k and t is one of the elements in the sequence $< tr[1], tr[2], \ldots >$, say $tr[j]$, then a mode change protocol will switch the mode of the tasks from mode k to mode $newmode[j]$. The tasks do not necessarily switch to the new mode immediately. Figure 1 shows that although the tasks are required to switch to $mode^1$ at time 100, task τ_2 switches to mode $mode^1$ a little bit later, namely at time 130 simply because it has to continue the execution of its current job before it can switch to the new mode. The general rule for mode change when the system is in steady state at time t is as follows. If task τ_i arrives at time $tr[j]$ then task τ_i switches from mode k to $newmode[j]$ immediately on its arrival; otherwise task τ_i switches from mode k to $newmode[j]$ at time $latest\_arrival(t, \tau_i) + T_i^k$.

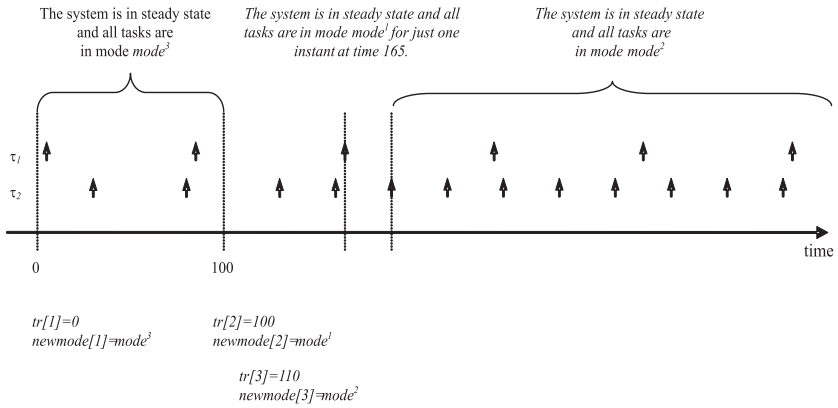

Fig. 2. An example showing the behavior of the system when a mode change is requested to be performed when the system is in a transient state. The example differs from the example in Figure 1 only in that a mode change is request at time 110 instead of at time 300.

Figure 2 illustrates another scenario showing the behavior of the system when a mode change is requested to be performed when the system is in a transient state. The example differs from the example in Figure 1 only in that a mode change is request at time 110 instead of at time 300.

The rule for the mode change protocol is as follows. If the system is in transient state at time t and the task set τ is in mode k and t is one of the elements in the sequence $< tr[1], tr[2], \ldots >$, say $tr[j]$, then no mode change is performed when the system is in a transient state; instead the tuple ($tr[j]$, $newmode[j]$) becomes member of the set *pending_mode_changes* and then immediately when the system enters steady state, the run-time system selects one (the application developer can choose which one) of the tuples in *pending_mode_changes* (let us say that ($tr[q]$, $newmode[q]$) was selected) and then acts as if it was requested that the system changes to mode $newmode[q]$ at the time when the system entered steady state. And then the set *pending_mode_changes* is assigned the empty set. The reason for this behavior is that we do not want to perform a mode change when the previously requested mode change has not yet been completed.

With this behavior in mind, let us revisit Figure 2. At time 100, it is requested that the system switches to mode1. Task τ_2 has to wait until time 130 until it can switch to $mode^1$ and task τ_1 has to wait until time 165 until it can switch to $mode^1$. But in the meantime, just a little bit later, it is requested that the system switches to $mode^2$. Therefore, the set pending_mode_changes becomes $\{mode^2\}$. At time 130, task τ_2 arrives again and hence switches to mode $mode^1$. But the system is still in transition state. At time 165, task τ_1 switches to $mode^1$ and then the system becomes in steady state. Immediately, when the system becomes in steady state, at time 165, the mode change request that were originally made at time 110 (because tr[3]=110) is requested now

(it is taken from pending_mode_changes) and the system acts as if there was a mode change request at time 165 with the new mode $mode^2$. Then the set $pending\_mode\_changes$ becomes the empty set.

We assume that $\forall i, k : 0 \leq C_i^k \leq T_i^k$ and C_i^k and T_i^k are real numbers. We also assume that task switching takes no time and a task needs no other resources than the processor.

The assumptions stated so far in this section are based on previously published work in the real-time literature [6, 7] with some clarification made but we also use the sporadic model which is more general than the periodic model used in [6, 7].

3 The utilization bound of EDF with mode change

In proofs, we will find it useful to discuss an algorithm called Processor-Sharing (PS). It operates as follows. Consider a time interval of duration $\epsilon > 0$ and assume that task τ_i is in the mode k during the entire time interval. Then it holds that τ_i executes for $(C_i^k/T_i^k) \cdot \epsilon$ time units during the time interval of duration ϵ.

Lemma 1. *Let $current\_modes(i, \tau)$ denote the set of modes of tasks in the task set τ at time t. It holds that $\forall t : current\_modes(i, \tau) \leq 2$.*

Proof. The lemma follows from the fact we do not (as stated in Section 2) allow a mode change when the system is in transient state.

Lemma 2. *If $\forall modes^k \in modes$ it holds that:*

$$\sum_{\tau_j \in \tau} \frac{C_j^k}{T_j^k} \leq \frac{1}{2} \tag{1}$$

and PS is used to schedule tasks then all deadlines are met.

Proof. From Lemma 1 and Equation 1 it follows that PS meets all deadlines.

Theorem 1. *If $\forall modes^k \in modes$ it holds that:*

$$\sum_{\tau_j \in \tau} \frac{C_j^k}{T_j^k} \leq \frac{1}{2} \tag{2}$$

and EDF is used to schedule tasks then all deadlines are met.

Proof. Follows from Lemma 2 and the fact that EDF is an optimal scheduling algorithm for a set of jobs [9]. (A scheduling algorithm is said to be optimal if it meets deadlines when it is possible to do so.)

The utilization bound expressed by Theorem 1 is tight; Example 1 shows that.

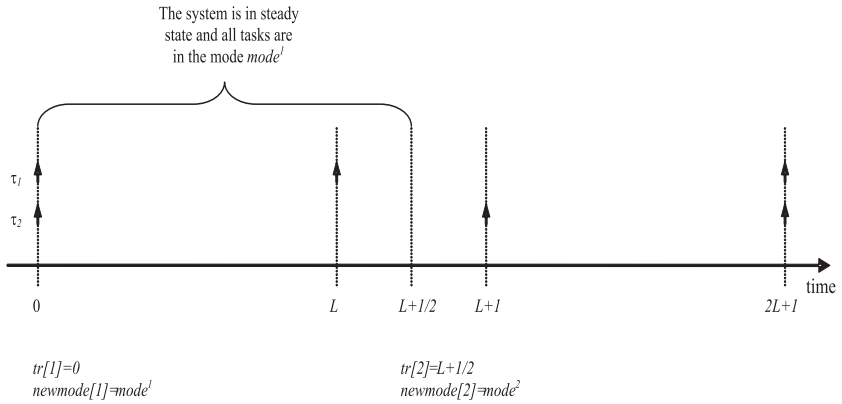

Fig. 3. An example showing the deadlines can be missed although the utilization exceeds 50% by just an arbitrarily small amount.

Example 1. The example is illustrated by Figure 3. Consider a task set $\tau = \{\tau_1, \tau_2\}$ with minimum inter-arrival times given as:

$T_1^1 = L \qquad T_1^2 = L + 1$
$T_2^1 = L + 1 \; T_2^2 = L$
and execution times given as

$C_1^1 = L/2 + 2 - L/(L+1) \; C_1^2 = 1$
$C_2^1 = 1 \qquad\qquad\qquad\qquad C_2^2 = L/2 + 2 - L/(L+1)$

where L is an integer ≥ 4. It is easy to verify that:

$$\sum_{\tau_j \in \tau} \frac{C_j^1}{T_j^1} = \sum_{\tau_j \in \tau} \frac{C_j^2}{T_j^2} = \frac{1}{2} + \frac{2}{L} \tag{3}$$

Figure 3 shows the run-time behavior. Both tasks are in $mode^1$ at time zero and both tasks release jobs at that time. At time L, the task τ_1 has arrived for the second time and the execution time of this job is $L/2 + 2 - L/(L+1)$ which is greater than 2. And hence the processor will be busy during the time interval $[L,L+2)$ and potentially longer. At time $L + 1/2$, the system is requested to change to $mode^2$; the processor is busy at this time. At time $L+1$, task τ_2 changes from $mode^1$ to $mode^2$. Task τ_1 must wait until it arrives again; therefore at time $2L + 1$, task τ_1 changes from $mode^1$ to $mode^2$. Let us calculate the amount of execution that must be performed during the time interval $[L + 1,2L + 1)$ in order to meet deadlines. Clearly, τ_2 needs to execute C_2^2 time units. τ_1 needs to execute $C_1^1 - 1$ time units. Therefore the two tasks τ_1 and τ_2 must execute

$C_2^2 + C_1^1 - 1$ during the time interval $[L+1,2L+1)$. That is, $L/2+2-L/(L+1)+L/2+2-L/(L+1)-1$ units of execution must be performed during a time interval of duration L. Rewriting gives us that $L+4-2L/(L+1)$ time units of execution must be performed during a time interval of duration L. Since $L \geq 4$, this is impossible and hence a deadline is missed.

We can repeat this argument for every $L \geq 4$. Letting $L \rightarrow \infty$ gives us that there exists a task set that misses a deadline although the utilization exceeds 50% by just an arbitrarily small amount.

4 Schedulability analysis of EDF with mode change

Although the utilization bound expressed by Theorem 1 is tight, there are task sets that can be scheduled with EDF under mode change although the utilization of a mode exceeds 50%. It is therefore of interest to design a schedulability analysis that can offer pre-run-time guarantees without checking the utilization.

The recurring task model [10] was developed to allow designers to model tasks with many blocks of instructions and the block of instructions selected for execution was depending on the outcome of an "if-statement". The model can be used for modeling task sets that undergo mode changes but only when the minimum inter-arrival time of tasks do not change across mode changes. For the case where minimum inter-arrival times are requested to be changed across mode changes, we are forced to develop a new schedulability analysis. We will study the special case where (i) |modes|=2 and (ii) T_i^1,C_i^1, T_i^2,C_i^2 and $tr[1]$ are integers and arrivals occur only at times which are integers and (iii) only one mode change request can occur during a busy interval. (A busy interval is an interval such that the processor is busy during this interval and just before the interval, the processor is idle and just after the interval, the processor is idle as well.) We believe this limitation is reasonable for systems where reconfiguration is performed not too often but when reconfiguration is required, the reconfiguration must be completed quickly, for example reconfiguration after the occurrence of a fault.

The notion of processor demand has played an important role in previous work [11] on schedulability analysis for uniprocessor EDF; we will use that notion in our analysis as well. Let R denote an assignment of arrival times to all jobs. Let $dbf(\ \tau_i,\ [t_0,t_1),\ R)$ denote the processor demand of all jobs released by task τ_i in the time interval $[t_0,t_1)$ for the assignment R, assuming that the assignment R satisfies the constraint of the minimum inter-arrival times of τ_i. More specifically, $dbf(\ \tau_i,\ [t_0,t_1),\ R)$ is defined as the sum of execution time of all jobs which satisfy the constraint of the minimum inter-arrival times of τ_i and that satisfy the following two conditions (i) the job arrives no earlier than t_0 and (ii) the deadline of the job is no later than t_1. Let $dbf(\ \tau_i,\ [t_0,t_1))$ denote the maximum of $dbf(\ \tau_i,\ [t_0,t_1),\ R)$ over all R such that R satisfies the constraint of the minimum inter-arrival times of τ_i. Let $dbf(\ \tau_i,\ L)$ denote the maximum of $dbf(\ \tau_i,\ [t_0,t_1))$ over all time intervals $[t_0,t_1)$ such that $t_1 - t_0 = L$. Finally, let us define $dbf(\ \tau,\ L)$ as:

$$dbf(\tau, L) = \sum_{\tau_j \in \tau} dbf(\tau_j, L) \tag{4}$$

The usefulness of the notion of processor demand follows from the fact [11] that:

> If jobs are released from tasks in a task set τ and jobs are scheduled with EDF and for every $L > 0$, it holds that if $dbf(\ \tau,\ L) \leq L$ then all deadlines are met.

This result was originally applied to sporadically arriving tasks [11] where equations were given on how to compute processor demand from task parameters. Such equations have also been developed for other task models, such as the multiframe model [12], the generalized multiframe model [13] and the recurring model [10]. Unfortunately, the research literature offers no such result for sporadic tasks with mode change. We will now address it.

Consider a time interval $[t_0,t_1)$ with $t_1 - t_0 = L$. The system is in $mode^1$ at time t_0 and at time $tr[1]$, there is a request to change to $newmode[1]$. Let $transition_j$ denote the time when task τ_j switches to mode $tr[1]$. We have that:

$$dbf(\tau, L) = \max_{t_0 \leq tr[1] \leq t_1} dbf(\tau, L, tr[1]) \tag{5}$$

where

$$dbf(\tau, L, tr[1]) = \sum_{\tau_j \in \tau} \left(\lfloor \frac{transition_j - t_0}{T_j^1} \rfloor \cdot C_j^1 + \lfloor \frac{t_1 - transition_j}{T_j^2} \rfloor \cdot C_j^2 \right) \tag{6}$$

and

$$\forall j : t_0 \leq transition_j \leq t_1 \tag{7}$$

and

$$\forall j : tr[1] \leq transition_j \tag{8}$$

and

$$\forall j : transition_j < tr[1] + T_j^1 \tag{9}$$

and

$$t_0 \leq tr[1] \leq t_1 \tag{10}$$

Intuitively, Inequality 5 states that the processor demand in a time interval of duration L is the maximum of all scenarios of times for requesting a mode change. Inequality 6 states that the processor demand of a task can be computed by adding the amount of execution before the task performs the transition and

the amount of execution after the task performs the transition. Inequality 7 states, for each task, that the transition occurs in the busy interval. Inequality 8 states that the transition of a task τ_j must not be earlier than the time when the mode change was requested and Inequality 9 states that when a task τ_j is in $mode^1$ and task τ_j is requested to perform a mode change, it will perform a mode change with a delay of at most T_j^1. Note that there is a strict inequality in Inequality 9. Inequality 10 states that the time when the mode change is requested is during the busy interval.

It is possible to compute $dbf(\tau, L)$ for a fixed L by solving the optimization problem expressed by Inequality 5 - Inequality 10. But recall that we need to calculate $dbf(\tau, L)$ for all positive values of L. For uniprocessor scheduling of EDF without mode change, it was shown [11] that if the utilization of the task set is known then one can find an upper bound on L such that values of L above this bound does not need to be checked. We will now develop a similar approach for EDF with mode change.

We will first state lemmas which express an upper bound on $dbf(\tau, L)$ (Lemma 3 and Lemma 4 do that) and then use these lemmas to derive an upper bound on the values of L that must be checked. (Lemma 5 does that).

Lemma 3. *It holds that:*

$$dbf(\tau, L, tr[1]) \leq \Big(\sum_{j\in\tau} C_j^1\Big) + L \cdot \max\Big(\sum_{j\in\tau} \frac{C_j^1}{T_j^1}, \sum_{j\in\tau} \frac{C_j^2}{T_j^2}\Big) \tag{11}$$

Proof. From Inequality 6,Inequality 8 and Inequality 9 it follows that:

$$dbf(\tau, L, tr[1]) \leq \sum_{\tau_j\in\tau} \Big(\lfloor \frac{tr[1] + T_j^1 - t_0}{T_j^1} \rfloor \cdot C_j^1 + \lfloor \frac{t_1 - tr[1]}{T_j^2} \rfloor \cdot C_j^2\Big) \tag{12}$$

For every $x > 0$ it holds that $\lfloor x \rfloor \leq x$. Using that on Inequality 12 gives us:

$$dbf(\tau, L, tr[1]) \leq \sum_{\tau_j\in\tau} \Big(\frac{tr[1] + T_j^1 - t_0}{T_j^1} \cdot C_j^1 + \frac{t_1 - tr[1]}{T_j^2} \cdot C_j^2\Big) \tag{13}$$

Using the fact that t_1=t_0+L on Inequality 13 yields:

$$dbf(\tau, L, tr[1]) \leq \sum_{\tau_j\in\tau} \Big(\frac{tr[1] + T_j^1 - t_0}{T_j^1} \cdot C_j^1 + \frac{t_0 + L - tr[1]}{T_j^2} \cdot C_j^2\Big) \tag{14}$$

Rewriting Inequality 14 yields:

$$dbf(\tau, L, tr[1]) \leq (tr[1] - t_0) \cdot \Big(\sum_{\tau_j\in\tau} (\frac{C_j^1}{T_j^1} - \frac{C_j^2}{T_j^2})\Big) + (\sum_{\tau_j\in\tau} C_j^1) + L \cdot (\sum_{\tau_j\in\tau} \frac{C_j^2}{T_j^2}) \tag{15}$$

Consider the term

$$\Big(\sum_{\tau_j \in \tau} (\frac{C_j^1}{T_j^1} - \frac{C_j^2}{T_j^2}) \Big) \tag{16}$$

If it is negative or zero then Inequality 15 is maximized for $tr[1] = t_0$ and hence for that case, an upper bound on Inequality 15 is:

$$(\sum_{\tau_j \in \tau} C_j^1) + L \cdot (\sum_{\tau_j \in \tau} \frac{C_j^2}{T_j^2}) \tag{17}$$

If Inequality 16 is positive then Inequality 15 is maximized for $tr[1] = t_0 + L$ and hence for that case, an upper bound on Inequality 15 is:

$$L \cdot \Big(\sum_{\tau_j \in \tau} (\frac{C_j^1}{T_j^1} - \frac{C_j^2}{T_j^2}) \Big) + (\sum_{\tau_j \in \tau} C_j^1) + L \cdot (\sum_{\tau_j \in \tau} \frac{C_j^2}{T_j^2}) \tag{18}$$

which can be rewritten to:

$$(\sum_{\tau_j \in \tau} C_j^1) + L \cdot (\sum_{\tau_j \in \tau} \frac{C_j^1}{T_j^1}) \tag{19}$$

Combining these cases gives us that:

$$dbf(\tau, L, tr[1]) \leq \Big(\sum_{j \in \tau} C_j^1 \Big) + L \cdot \max \Big(\sum_{j \in \tau} \frac{C_j^1}{T_j^1}, \sum_{j \in \tau} \frac{C_j^2}{T_j^2} \Big) \tag{20}$$

This states the lemma.

Lemma 4. *It holds that:*

$$dbf(\tau, L) \leq \Big(\sum_{j \in \tau} C_j^1 \Big) + L \cdot \max \Big(\sum_{j \in \tau} \frac{C_j^1}{T_j^1}, \sum_{j \in \tau} \frac{C_j^2}{T_j^2} \Big) \tag{21}$$

Proof. Follows from Inequality 5 and Lemma 3.

Lemma 5. *If*

$$\max \Big(\sum_{j \in \tau} \frac{C_j^1}{T_j^1}, \sum_{j \in \tau} \frac{C_j^2}{T_j^2} \Big) < 1 \tag{22}$$

then for all $L > 0$ such that

$$\frac{\sum_{j \in \tau} C_j^1}{1 - \max \Big(\sum_{j \in \tau} \frac{C_j^1}{T_j^1}, \sum_{j \in \tau} \frac{C_j^2}{T_j^2} \Big)} \leq L \tag{23}$$

it holds that:

$$dbf(\tau, L) \leq L \tag{24}$$

Proof. Rewriting Inequality 23 gives us:

$$\Big(\sum_{j\in\tau} C_j^1\Big) + L \cdot \max\Big(\sum_{j\in\tau} \frac{C_j^1}{T_j^1}, \sum_{j\in\tau} \frac{C_j^2}{T_j^2}\Big) \leq L \tag{25}$$

Using Lemma 4 on Inequality 25 gives us:

$$dbf(\tau, L) \leq L \tag{26}$$

This states the lemma.

We now know that only values of L that do not exceed the left-hand side of Inequality 23 must be checked. We can enumerate all values of L from 1 up to this bound. We can also assume (with no loss of generality) that $t_0 = 0$ and $t_1 = L$ and hence enumerate $tr[1]$ from 0 up to L. For each of these we would like to compute $dbf(\ \tau, L, tr[1])$, using Inequality 6. This can be done by iterating, for each task τ_i, through all values of $transition_j$ from $tr[1]$ to $tr[1] + T_j^1 - 1$ and compute the term in the sum of Inequality 6; for convenience this term is stated below.

$$\lfloor \frac{transition_j - t_0}{T_j^1} \rfloor \cdot C_j^1 + \lfloor \frac{t_1 - transition_j}{T_j^2} \rfloor \cdot C_j^2 \tag{27}$$

Based on these ideas we obtain the schedulability test expressed by Figure 4.

5 Conclusions

We have presented the first result on uniprocessor EDF scheduling with mode change. We used the previously known, and well established, mode-change protocol designed by Sha et al. [6]. The schedulability analysis is simple; if the utilization of every mode is at most 50% then all deadlines are met. We also show that there exists a task set that misses a deadline although the utilization exceeds 50% by just an arbitrarily small amount. Finally, we presented, for a relevant special case, an exact schedulability test for EDF with mode change. We left open the problem of designing an exact schedulability test for the case where minimum inter-arrival times and execution times are real numbers.

Acknowledgements

This work was partially funded by the Portuguese Science and Technology Foundation (Fundação para a Ciência e a Tecnologia - FCT) and the ARTIST2 Network of Excellence on Embedded Systems Design.

References

[1] Lee, C.-G., Shih, C.-S., Sha, L: Service class based online QoS management in surveillance radar systems. IEEE Real-Time Systems Symposium, London, UK, 2001.
[2] Rajkumar, R. Lee, C. Lehoczky, J. Siewiorek, D.: A Resource Allocation Model for QoS Management. Proceedings of the IEEE Real-Time Systems Symposium, San Francisco, CA, USA, 1997.
[3] Rosu, D., Schwan, K., Yalamanchili, S., Jha, R.: On adaptive resource allocation for complex real-time application. Proceedings of the IEEE Real-Time Systems Symposium, San Francisco, CA, USA, 1997.
[4] Abdelzaher, T.F., Atkins, E.M., Shin, K.: QoS Negotiation in Real-Time Systems and its Application to Automated Flight Control. IEEE Transactions on Computers, vol. 49, 2000.
[5] Real, J. and Crespo, A.: Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal. Real-Time Systems, vol. 26, pp. 161 - 197, 2004.
[6] Sha, L. Rajkumar, R. Lehoczky, J. and Ramamritham, K.: Mode Change Protocol for Priority-Driven Preemptive Scheduling. Carnegie-Mellon University Pittsburgh, Pennsylvania, Software Engineering Institute November 1988.
[7] Tindell, K. Burns, A. Wellings, A.J.: Mode Changes In Priority Pre-Emptively Scheduled Systems. IEEE Real-Time Systems Symposium, Phoenix, Arizona, USA, 1992.
[8] Liu, C. L., Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Technical Report, Department of Computer Science, University of York, UK YCS 189., 1992. Journal of the ACM, vol. 20, pp. 46 - 61, 1973.
[9] Dertouzos, M.L: The Procedural Control of Physical Processes. IFIP Congress, Stockholm, Sweden, 1974.
[10] Baruah, S.: Dynamic- and static-priority scheduling of recurring real-time tasks. Real-time Systems, vol. 24, pp. 93-128, 2003.
[11] Baruah, S. Howell, R., Rosier, L.: Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. Real-Time Systems, vol. 2, pp. 301-324, 1990.
[12] Mok, A.K, Chen, D.: A multiframe model for real-time tasks. IEEE Real-Time Systems Symposium, Washington, DC, USA, 1996.
[13] Baruah, S. Chen, D., Gorinsky, S. Mok, A.: Generalized multiframe tasks. Real-Time Systems, vol. 17, pp. 5-22, 1999.

Input: a task set τ
Output: "unschedulable", "cannot decide" or "schedulable"
Assumptions:
- |modes|=2 and
- T_i^1,C_i^1, T_i^2,C_i^2 and $tr[1]$ are integers and arrivals occur only at times which are integers and only one mode change request can occur during a busy interval.
- only one mode change request can occur during a busy interval.

```
1.  Let U denote the left-hand side of (22)
2.  if U>1 then
3.      declare "unschedulable"
4.  else
5.      if U=1 then
6.          declare "cannot decide"
7.      else
8.          UBL := left-hand side of (23)
9.          for L in 1..UBL do
10.             t0 := 0
11.             t1 := L
12.             for tr[1] in 0..L do
13.                 sum := 0
14.                 for ∀τj ∈ τ do
15.                     sumj := 0
16.                     for transitionj in tr[1]..min(t1,tr[1]+Tj^1-1) do
17.                         if evaluation of (27) > sumj then
18.                             sumj := evaluation of (27)
19.                         end if
20.                     end for
21.                     sum := sum + sumj
22.                 end for
23.                 if sum>L then
24.                     declare "unschedulable"
25.                 end if
26.             end for
27.         end for
28.         declare "schedulable"
29.     end if
30. end if
```

Fig. 4. A schedulability test for EDF for checking whether changing from $mode^1$ to $mode^2$ can be performed and meet deadlines.