

# Flexible Tardiness Bounds for Sporadic Real-Time Task Systems on Multiprocessors \*

UmaMaheswari C. Devi and James H. Anderson  
Department of Computer Science  
The University of North Carolina at Chapel Hill

October, 2005

## Abstract

The earliest-deadline-first (EDF) scheduling of a sporadic real-time task system on a multiprocessor may require that the total utilization of the task system,  $U_{sum}$ , not exceed  $(m + 1)/2$  on  $m$  processors if every deadline needs to be met. In recent work, we considered the alleviation of this under-utilization for task systems that can tolerate deadline misses by bounded amounts (*i.e.*, bounded tardiness). We showed that if  $U_{sum} \leq m$  and tasks are not pinned to processors, then the tardiness of each task is bounded under both preemptive and non-preemptive EDF. The tardiness bounds that we derived are dependent upon the utilizations and execution costs of the constituent tasks, but are independent of  $U_{sum}$ . Furthermore, any task may incur maximum tardiness. In this paper, we address the issue of supporting tasks whose tolerance to tardiness is less than that known to be possible under EDF. We propose a new scheduling policy, called EDF-hl, that is a variant of EDF, and show that under EDF-hl, any tardiness, including zero tardiness, can be ensured for a limited number of *privileged* tasks, and that bounded tardiness can be guaranteed to the remaining tasks if their utilizations are restricted. EDF-hl reduces to EDF in the absence of privileged tasks. The tardiness bound that we derive is a function of  $U_{sum}$ , in addition to individual task parameters. Hence, tardiness for all tasks can be lowered by lowering  $U_{sum}$ . An experimental evaluation of the tardiness bounds that are possible is provided.

---

\*Work supported by NSF grants CCR 0309825 and CCR 0408996. The first author was also supported by an IBM Ph.D. fellowship.

# 1 Introduction

A real-time system has to meet certain *timing constraints* to be correct. Such timing constraints are typically specified as deadline requirements. Tasks in a real-time system are often recurrent in nature. The sporadic task model is one of the most widely-studied notions of recurrent real-time task execution. In this model, each task is a sequential program that is invoked repeatedly; each such invocation is called a *job*. The  $i^{\text{th}}$  task is denoted  $T_i(e_i, p_i)$ , where  $p_i > 0$  is the minimum *inter-arrival separation* for its successive jobs (*i.e.*, successive job invocations of  $T_i$  must be spaced apart by at least  $p_i$  time units), and  $e_i \leq p_i$  is its *per-job execution cost*.  $p_i$  is also referred to as the *period* of  $T_i$ . In the variant of the sporadic model considered here,  $p_i$  is also the *relative deadline* of  $T_i$ , *i.e.*, each job of  $T_i$  must complete execution within  $p_i$  time units of its invocation. The quantity  $e_i/p_i$  denotes the *utilization* of  $T_i$ . This quantity corresponds to the share of a single processor that  $T_i$  requires in the long run.

It is highly desirable that jobs be scheduled so that they do not miss their deadlines. However, in a *soft real-time system*, deadline misses can sometimes be tolerated, if the amount by which a deadline is missed is within a specified per-task *tardiness threshold*: If  $\delta$  is the tardiness threshold of task  $T_i$ , then a job of  $T_i$  with a deadline at time  $d$  should be guaranteed to complete execution by time  $d + \delta$ . Such a guarantee would ensure that each task receives a processor share close to its utilization.

In work on real-time systems, multiprocessor platforms (SMPs) are of growing importance. This is due to both hardware trends such as the emergence of multicore technologies, and also to the prevalence of computationally-intensive applications for which single-processor designs are not sufficient. Examples of such applications include systems that track people and machines, many computer-vision systems, and signal-processing applications such as synthetic aperture imaging (to name a few). Timing constraints in several of these applications are predominantly soft. Given these observations, designing efficient scheduling algo-

gorithms for multiprocessor-based soft real-time systems and extending the analysis of traditional algorithms to soft real-time systems are goals of considerable value and interest.

Sporadic task systems can be scheduled on a multiprocessor using either a *partitioning* or a *global-scheduling* approach. Under partitioning, tasks are statically assigned to processors, and a uniprocessor scheduling algorithm is used on each processor to schedule its assigned tasks. In contrast, under global scheduling, a task may execute on any processor and may migrate across processors. Each approach can be differentiated further based on the scheduling algorithm that is used. For instance, the *earliest-deadline-first* (EDF) [7] or the *rate-monotonic* (RM)\* [9] algorithm could be used as the per-processor scheduler under partitioning, or as the system-wide global scheduler.

Pfair scheduling [4], when deployed in a global setting, is currently the only known way of *optimally* scheduling sporadic task systems on a multiprocessor. (The term “optimal” means that such algorithms are capable of scheduling on  $m$  processors any task system with total utilization at most  $m$ .) However, Pfair algorithms schedule tasks one quantum at a time, and as a result, jobs may be preempted and migrate across processors frequently. Such preemption and migration overheads can lower the amount of useful work that is actually accomplished. On the other hand, no known non-Pfair-based scheduling algorithm is optimal, and in the worst case, every such algorithm requires that total utilization not exceed  $(m + 1)/2$  (*i.e.*, the underlying platform is underutilized by roughly 50%), if every deadline is to be met [5, 10, 3, 2].

Prior work has shown that such restrictions on overall utilization can be eliminated for soft real-time systems. In [1], Anderson *et al.* presented a variant of partitioned-EDF that ensures bounded tardiness with no such restrictions, provided per-task utilizations are capped at  $1/2$ . In addition, in a recent paper [8], we derived tardiness bounds for

---

\*Under RM scheduling, priorities for jobs are fixed offline and are inversely proportional to the periods of their tasks: the jobs of a task with a smaller period have higher priority than those of another task with a larger period.

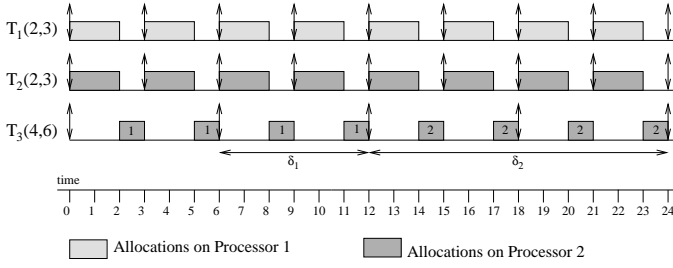


Figure 1: A global-RM schedule for an example task set. Numbers within shaded rectangles indicate job numbers.  $\delta_i$  indicates the tardiness of the  $i^{th}$  job of  $T_3$ .

both preemptive and non-preemptive global EDF. For EDF, we established a bound of  $((m-1)e_{\max} - e_{\min}) / (m - (m-2)u_{\max}) + e_{\min}$ , where  $m$  is the number of processors and  $e_{\max}$  (resp.,  $e_{\min}$ ) is the maximum (resp., minimum) execution cost of any task in the task system. For non-preemptive EDF, we established a bound of  $(me_{\max} - e_{\min}) / (m - (m-1)u_{\max}) + e_{\min}$ . Tardiness bounds under EDF have also been presented by Valente and Lipari [12]. A precursor to all of the work mentioned here is a paper by Srinivasan and Anderson [11] in which tardiness bounds are presented for the earliest-pseudo-deadline first (EPDF) Pfair scheduling algorithm, which is sub-optimal but more efficient than optimal algorithms.

While it may be reasonable to expect that most scheduling algorithms should be capable of guaranteeing bounded tardiness in the absence of overutilization, in reality this is not the case for some well-known algorithms. Partitioning algorithms and global RM are examples. Under partitioned scheduling, if a task set cannot be partitioned without overutilizing some processor, then tardiness for the tasks on that processor will increase with time. An example of a task system with unbounded tardiness under global RM is shown in Fig. 1. Assume here that each job of every task is invoked as early as permissible. Then, the first job of  $T_3$  does not complete executing until time 12, for a tardiness of 6 time units, and the second job suffers from a tardiness of 12 time units. It is easy to see that the  $i^{th}$  job of  $T_3$  does not complete until time  $12i$ , for all  $i$ , for a tardiness of  $6i$  time units. This tardiness increases with time and thus is unbounded.

**Contributions.** The tardiness bounds derived by us previously for preemptive and non-preemptive EDF [8] are dependent on per-task utilizations and execution costs but independent of total system utilization,  $U_{sum}$ . Furthermore, any task may incur maximum tardiness. This may not be acceptable to applications that are comprised of hard and soft real-time tasks or those comprised of soft tasks with different tardiness tolerances. In this paper, we make an attempt to address this limitation.

Our contributions are twofold. First, we consider guaranteeing lower tardiness to some tasks at the expense of others. To this end, we propose a new scheduling policy, called EDF-hl, which is a variant of global EDF. We show that under EDF-hl, on  $m$  processors, up to  $m$  tasks can be accorded preferential treatment and thereby guaranteed any tardiness, including zero, and that bounded tardiness can be guaranteed to the remaining tasks if their utilizations are capped. In the absence of tasks that require lower tardiness, EDF-hl reduces to EDF. Simulations involving randomly-generated task sets presented herein suggest that for many systems, the tardiness bounds that can be ensured for tasks that do not receive preferential treatment are acceptable.

Unlike in [8], the tardiness bound derived here is a function of  $U_{sum}$ , in addition to individual task parameters. Thus, as a second contribution, our bound offers the possibility of lowering tardiness for all tasks by lowering  $U_{sum}$ . To assess the tardiness-utilization trade-off for EDF (*i.e.*, without special tasks), we again conducted experiments involving randomly-generated task sets. We found that, with the improved analysis, considerable reductions in tardiness are possible even for reasonable reductions in total system utilization. For instance, in the simulation results for eight processors shown in Fig. 6(b) in Sec. 4, lowering  $U_{sum}$  by around 10% results in a reduction in maximum tardiness by over 35%, and lowering  $U_{sum}$  by 25% lowers maximum tardiness by close to 50%.

**Organization.** The rest of this paper is organized as follows. Our system model and algorithm EDF-hl are described in Sec. 2. Tardiness bounds are

derived in Sec. 3. An experimental evaluation of the tardiness-total utilization trade-off is provided in Sec. 4. Finally, Sec. 5 concludes.

## 2 Definitions

In this section, our task model is described and algorithm EDF-hl is presented.

**Task model.** A sporadic task system comprised of  $n \geq 1$  sporadic tasks is to be scheduled on  $m \geq 2$  processors. Each sporadic task  $T_i(e_i, p_i)$  is as described in the introduction. The *utilization* of  $T_i$  is given by  $u_i = e_i/p_i$ .  $u_i \leq 1$  holds for all  $i$ . The *total utilization* of  $\tau$  is defined as  $U_{sum}(\tau) = \sum_{i=1}^n u_i$ . It is required that  $U_{sum}(\tau) \leq m$  hold. The maximum utilization (resp., execution cost) of any task in  $\tau$  is denoted  $u_{max}(\tau)$  (resp.,  $e_{max}(\tau)$ ). The minimum execution cost of any task is denoted  $e_{min}(\tau)$ .  $\mathcal{U}_{max}(\tau, k)$ , where  $k \leq n$ , denotes the  $k$  tasks with highest utilizations in  $\tau$ . More formally,  $\mathcal{U}_{max}(\tau, k)$  denotes a subset of  $k$  tasks of  $\tau$ , where the utilization of each task in the subset is at least as high as that of every task in  $\tau \setminus \mathcal{U}_{max}(\tau, k)$ .  $\mathcal{E}_{max}(\tau, k)$  is defined analogously with respect to execution costs. (In all of these max and min terms,  $\tau$  will be omitted when it is unambiguous.)

The  $k^{th}$  job of  $T_i$ , where  $k \geq 1$ , is denoted  $T_{i,k}$ , and its *release time* and *absolute deadline* (or simply *deadline* for short) are denoted  $r_{i,k}$  and  $d_{i,k}(= r_{i,k} + p_i)$ , respectively.  $r_{i,k}$  denotes the invocation time of  $T_{i,k}$  and is the time at or after which  $T_{i,k}$  can be executed.  $r_{i,k+1} - r_{i,k} \geq p_i$  holds for all  $k \geq 1$ . Each task is sequential, and hence no job of any task may execute in parallel. Furthermore, no two jobs of any task may execute in parallel.

A sporadic task system  $\tau$  is said to be *concrete* if the release time of every job of each of its tasks is specified, and *non-concrete*, otherwise. Note that an infinite number of concrete task systems can be specified for every non-concrete task system. We omit specifying the type of the task system unless it is necessary. The results in this paper are for non-concrete task systems, and hence hold for every concrete task system.

The *tardiness of a job*  $T_{i,j}$  in a schedule  $\mathcal{S}$  is defined as  $tardiness(T_{i,j}, \mathcal{S}) = \max(0, t - d_{i,j})$ , where  $t$  is the time at which  $T_{i,j}$  completes executing in  $\mathcal{S}$ . The *tardiness of a task system*  $\tau$  under scheduling algorithm  $\mathcal{A}$  is defined as the maximum tardiness of any job of a task in  $\tau$  in any schedule under  $\mathcal{A}$ . If  $\kappa$  is the maximum tardiness of any task system under  $\mathcal{A}$ , then  $\mathcal{A}$  is said to *ensure a tardiness bound of  $\kappa$* . Though tasks in a soft real-time system are allowed to have nonzero tardiness, we assume that *missed deadlines do not delay future job releases*. That is, even if a job of a task misses its deadline, the release time of the next job of that task remains unaltered. Since consecutive jobs of the same task cannot be scheduled in parallel, a missed deadline effectively reduces the interval over which the next job should be scheduled in order to meet its deadline.

The sporadic task model is augmented as follows for EDF-hl (described below). Each task in  $\tau$  is classified as either a *privileged task* or an *unprivileged task*. The set of all privileged (resp., unprivileged) tasks is denoted  $\tau_H$  (resp.,  $\tau_L$ ). ( $H$  and  $L$  stand for high and low privilege, respectively.)  $|\tau_H| \leq m$  holds. Each privileged task  $T_h$  has a maximum tardiness parameter  $\Delta_h \geq 0$ , which denotes the maximum tardiness that any of its jobs can tolerate.  $d_{h,j} + \Delta_h$  is referred to as the *effective deadline* of job  $T_{h,j}$  and is denoted  $\rho_{h,j}$ .

**Algorithm EDF-hl.** Our goal is to design an algorithm that can guarantee a tardiness of  $\Delta_h$  to each privileged task  $T_h$  while guaranteeing bounded tardiness to the remaining tasks. Let the *slack* of a job  $T_{h,j}$  of a privileged task  $T_h$  at time  $t$  be defined as  $d_{h,j} + \Delta_h - t - (e_h - \delta_{h,j})$ , where  $\delta_{h,j}$  denotes the amount of time that  $T_{h,j}$  executed before  $t$ . Informally, the slack of job  $T_{h,j}$  at  $t$  is the amount of time it can afford *not* to execute after  $t$  until completion for its tardiness to be at most  $\Delta_h$ . A tardiness of at most  $\Delta_h$  can be guaranteed for task  $T_h$  if each job  $T_{h,j}$  is scheduled based on its deadline until time  $d_{h,j} + \Delta_h - e_h$ , but is guaranteed continuous execution from  $d_{h,j} + \Delta_h - e_h$  onward. (This is somewhat similar to the behavior of the earliest-deadline-until-zero-laxity algorithm

described in [6].) Job  $T_{h,j}$  is said to be *urgent* at time  $t$ , if  $t \geq d_{h,j} + \Delta_h - e_h$  and  $T_{h,j}$  has not completed execution by  $t$ . Note that  $T_{h,j}$  is flagged as urgent from  $d_{h,j} + \Delta_h - e_h$  until completion even if its slack is positive. This eliminates the overhead of updating the urgency for each privileged job at runtime and may also result in fewer preemptions and migrations.

With the above definitions in place, Algorithm EDF-hl can be described as follows. At any time  $t$ , each of the urgent jobs, if any, of tasks in  $\tau_H$  is assigned a unique processor. If not every processor is assigned to an urgent job, then the non-urgent jobs of  $\tau_H$  and jobs of tasks in  $\tau_L$  are scheduled on the remaining processors on an earliest-deadline-first basis, where ties are resolved arbitrarily. A job may be preempted at any time by a higher priority job and may later resume execution on a different processor.

Note that EDF-hl reduces to EDF if  $\tau_H = \emptyset$ . Since  $|\tau_H| \leq m$  holds, EDF-hl clearly ensures the required tardiness for each privileged task. Hence, the question to be addressed is whether bounded tardiness can be guaranteed for the remaining tasks. The answer turns out to be yes if there is a cap on the utilizations of the remaining tasks. This cap depends on the number of privileged tasks and their utilizations. To see that such a cap is necessary, at least in some cases, consider a task system comprised of four tasks  $T_1(3, 4), \dots, T_3(3, 4)$ , and  $T_4(3i, 4i)$ , where  $i \geq 1$ . If tasks  $T_1, \dots, T_3$  require a tardiness of zero, then tardiness for  $T_4$  can grow unboundedly.

**Discussion.** Though the tardiness bounds derived in [8] guarantee that tardiness for each task in the above example (with  $i = 1$ ) is at most 4.33 time units under EDF, no task is immune from incurring maximum tardiness. The bound for EDF-hl derived here would enable one of the four tasks to be guaranteed zero tardiness if the remaining tasks can tolerate a tardiness of 6 time units (which is only slightly higher than 4.33). However, if two tasks have a tardiness requirement of zero, then tardiness for the remaining tasks may be as high as 21.0 (which is still bounded). Lower tardiness can

be guaranteed if the utilizations of the unprivileged tasks are lower. For instance, with two privileged tasks  $T_1(3, 4)$  and  $T_2(3, 4)$  and three unprivileged tasks  $T_3(3, 6), \dots, T_5(3, 6)$ , the unprivileged tasks would have a tardiness of at most 12.0.

### 3 Tardiness under EDF-hl

In this section, we determine a tardiness bound for  $\tau_L$ . The approach for doing this is the same as that used in [8]. This involves comparing the allocations to a concrete task system  $\tau$  in a processor sharing (PS) schedule for  $\tau$  and an actual EDF-hl schedule of interest for  $\tau$ , and quantifying the difference between the two. In a PS schedule, each job of  $T_i$  is allocated a fraction  $u_i$  of a processor at each instant (or equivalently, a fraction  $u_i$  of each instant) in the interval between its release time and its deadline. Because  $U_{sum} \leq m$  holds, the total demand at any instant will not exceed  $m$  in a PS schedule, and hence no deadlines will be missed; in fact, every job will complete executing exactly at its deadline. We begin by setting the required machinery in place.

#### 3.1 Definitions and Notation

A time interval  $[t_1, t_2)$ , where  $t_2 \geq t_1$ , consists of all times  $t$ , where  $t_1 \leq t < t_2$ , and is of length  $t_2 - t_1$ . The system start time is assumed to be zero. For any time  $t > 0$ ,  $t^-$  denotes the time  $t - \epsilon$  in the limit  $\epsilon \rightarrow 0+$ .

**Definition 1 (active tasks and active jobs):** A task  $T_i$  is said to be *active* at time  $t$ , if there exists a job  $T_{i,j}$  (called  $T_i$ 's *active job* at  $t$ ) such that  $r_{i,j} \leq t < d_{i,j}$ . By our task model, every task can have at most one active job at any time.

**Definition 2 (pending jobs):**  $T_{i,j}$  is said to be *pending* at  $t$  in a schedule  $\mathcal{S}$  if  $r_{i,j} \leq t$  and  $T_{i,j}$  has not completed execution by  $t$  in  $\mathcal{S}$ . Note that a job with a deadline at or before  $t$  is not considered to be active at  $t$  even if it is pending at  $t$ .

**Definition 3 (ready jobs):** A pending job  $T_{i,j}$  is said to be *ready* at  $t$  in a schedule  $\mathcal{S}$  if  $t \geq r_{i,j}$  and

all prior jobs of  $T_i$  have completed execution by  $t$  in  $\mathcal{S}$ .

We now quantify the total allocation to  $\tau$  in an interval  $[t_1, t_2]$  in a PS schedule for  $\tau$ ,  $\text{PS}_\tau$ . Let  $A(\mathcal{S}, T_i, t_1, t_2)$  denote the total time allocated to  $T_i$  in an arbitrary schedule  $\mathcal{S}$  for  $\tau$  in  $[t_1, t_2]$ . Then, since  $T_i$  is allocated in  $\text{PS}_\tau$  a fraction  $u_i$  of each instant at which it is active in  $[t_1, t_2]$ , we have

$$A(\text{PS}_\tau, T_i, t_1, t_2) \leq (t_2 - t_1)u_i. \quad (1)$$

The total allocation to  $\tau$  in the same interval in  $\text{PS}_\tau$  is

$$\begin{aligned} & A(\text{PS}_\tau, \tau, t_1, t_2) \\ & \leq \sum_{T_i \in \tau} (t_2 - t_1)u_i = U_{sum}(\tau) \cdot (t_2 - t_1). \end{aligned} \quad (2)$$

We are now ready to define lag and LAG, which play a pivotal role in this paper. The *lag of task  $T_i$  at time  $t$  in schedule  $\mathcal{S}$* , denoted  $\text{lag}(T_i, t, \mathcal{S})$ , is given by

$$\text{lag}(T_i, t, \mathcal{S}) = A(\text{PS}_\tau, T_i, 0, t) - A(\mathcal{S}, T_i, 0, t). \quad (3)$$

In  $\mathcal{S}$ , less work than in  $\text{PS}_\tau$  on the jobs of  $T_i$  has been completed by time  $t$  if  $\text{lag}(T_i, t, \mathcal{S})$  is positive (*i.e.*,  $T_i$  is under-allocated in  $\mathcal{S}$ ), and more work, if  $\text{lag}(T_i, t, \mathcal{S})$  is negative (*i.e.*,  $T_i$  is over-allocated in  $\mathcal{S}$ ). The *total lag of a task system  $\tau$  at  $t$* , denoted  $\text{LAG}(\tau, t, \mathcal{S})$ , is given by

$$\begin{aligned} \text{LAG}(\tau, t, \mathcal{S}) &= \sum_{T_i \in \tau} \text{lag}(T_i, t, \mathcal{S}) \\ &= A(\text{PS}_\tau, \tau, 0, t) - A(\mathcal{S}, \tau, 0, t). \end{aligned} \quad (4)$$

Note that  $\text{LAG}(\tau, 0, \mathcal{S})$  and  $\text{lag}(T_i, 0, \mathcal{S})$  are both zero, and that by (3) and (4), we have the following for  $t_2 > t_1$ .

$$\begin{aligned} \text{lag}(T_i, t_2, \mathcal{S}) &= \text{lag}(T_i, t_1, \mathcal{S}) + \\ & A(\text{PS}_\tau, T_i, t_1, t_2) - A(\mathcal{S}, T_i, t_1, t_2) \end{aligned} \quad (5)$$

$$\begin{aligned} \text{LAG}(\tau, t_2, \mathcal{S}) &= \text{LAG}(\tau, t_1, \mathcal{S}) + \\ & A(\text{PS}_\tau, \tau, t_1, t_2) - A(\mathcal{S}, \tau, t_1, t_2) \end{aligned} \quad (6)$$

**Lag for jobs.** The notion of lag defined above for tasks and task sets can be applied to jobs and job sets in an obvious manner. Let  $\tau$  denote a concrete task system, and  $\Psi$  a subset of jobs in

$\tau$ . Let  $A(\text{PS}_\tau, T_{i,j}, t_1, t_2)$  and  $A(\mathcal{S}, T_{i,j}, t_1, t_2)$  denote the allocations to  $T_{i,j}$  in  $[t_1, t_2]$  in  $\text{PS}_\tau$  and  $\mathcal{S}$ , respectively. Then,  $\text{lag}(T_{i,j}, t, \mathcal{S}) = A(\text{PS}_\tau, T_{i,j}, r_{i,j}, t) - A(\mathcal{S}, T_{i,j}, r_{i,j}, t)$ , and  $\text{LAG}(\Psi, t, \mathcal{S}) = \sum_{T_{i,j} \in \Psi} \text{lag}(T_{i,j}, t, \mathcal{S})$ . The total allocation in  $[0, t]$ , where  $t > 0$ , to a job that is neither pending at  $t^-$  in  $\mathcal{S}$  nor is active at  $t^-$  is the same in both  $\mathcal{S}$  and  $\text{PS}_\tau$ , and hence, its lag at  $t$  is zero. Therefore, for  $t > 0$ , we have

$$\begin{aligned} \text{LAG}(\Psi, t, \mathcal{S}) &= \sum_{\substack{T_{i,j} \text{ is in } \Psi, \text{ and is} \\ \text{pending or active at} \\ t^-}} \text{lag}(T_{i,j}, t, \mathcal{S}). \end{aligned}$$

The above expression can be rewritten using task lags as follows (since no job can be scheduled before its release time).

$$\begin{aligned} \text{LAG}(\Psi, t, \mathcal{S}) &\leq \sum_{\substack{T_i \in \tau : T_{i,j} \text{ is in } \Psi, \\ \text{and is pending or active} \\ \text{at } t^-}} \text{lag}(T_i, t, \mathcal{S}) \end{aligned} \quad (7)$$

Similarly, the total utilization of  $\Psi$  at time  $t$  is given by the sum of the utilizations of tasks with an active job at  $t$  in  $\Psi$ :

$$\begin{aligned} U_{sum}(\Psi, t) &= \sum_{\substack{T_i \in \tau : T_{i,j} \text{ is in } \Psi \text{ and} \\ \text{is active at } t}} u_i. \end{aligned} \quad (8)$$

**Definition 4 (busy interval):** A time interval  $[t_1, t_2)$ , where  $t_2 > t_1$ , is said to be *busy* for  $\tau$  if all  $m$  processors are executing jobs of tasks in  $\tau$  throughout the interval, *i.e.*, no processor is ever idle in the interval or executes a job of a task not in  $\tau$ . An interval  $[t_1, t_2)$  that is not busy for  $\tau$  is said to be *non-busy* for  $\tau$ , and is *maximally non-busy* if every time instant in  $[t_1, t_2)$  is non-busy, and either  $t_1 = 0$  or  $t_1^-$  is busy.

If at least  $U_{sum}(\tau)$  tasks are executing at any instant in  $[t_1, t_2)$  in a schedule  $\mathcal{S}$  for  $\tau$ , then the tasks in  $\tau$  receive a total allocation of  $U_{sum}(\tau) \cdot (t_2 - t_1)$  time in  $\mathcal{S}$  in that interval. By (2), the total allocation to  $\tau$  in  $[t_1, t_2)$  cannot exceed  $U_{sum}(\tau) \cdot (t_2 - t_1)$  in  $\text{PS}_\tau$ . Therefore, by (6), the LAG of  $\tau$  at  $t_2$  cannot exceed that at  $t_1$ , and we have the following lemma.

**Lemma 1** *If  $\text{LAG}(\tau, t + \delta, \mathcal{S}) > \text{LAG}(\tau, t, \mathcal{S})$ , where  $\delta > 0$  and  $\mathcal{S}$  is a schedule for  $\tau$ , then  $[t, t + \delta)$  is a non-busy interval for  $\tau$ . Furthermore, there exists at least one instant in  $[t, t + \delta)$  at which fewer than  $U_{\text{sum}}(\tau)$  tasks are executing.*

The busy interval in Def. 4 is defined with respect to  $\tau$ . With respect to  $\Psi$ ,  $[t_1, t_2)$  is said to be *busy* only if every processor is executing some job of  $\Psi$  throughout  $[t_1, t_2)$ . The job-set counterpart of Lemma 1 is as follows.

**Lemma 2** *If  $\text{LAG}(\Psi, t + \delta, \mathcal{S}) > \text{LAG}(\Psi, t, \mathcal{S})$ , where  $\delta > 0$  and  $\mathcal{S}$  is a schedule for  $\Psi$ , then  $[t, t + \delta)$  is a non-busy interval for  $\Psi$ . Furthermore, there exists at least one instant  $t'$  in  $[t, t + \delta)$  at which fewer than  $U_{\text{sum}}(\Psi, t')$  tasks are executing jobs in  $\Psi$ .*

## 3.2 Deriving a Tardiness Bound

Given an arbitrary non-concrete task system  $\tau^N$ , we are interested in determining the highest tardiness of any job of any task in  $\tau_L^N$  in any concrete instantiation of  $\tau^N$ . Let  $\tau$  (resp.,  $\tau_H$  and  $\tau_L$ ) be a concrete instantiation of  $\tau^N$  (resp.,  $\tau_H^N$  and  $\tau_L^N$ ),  $T_{\ell,j}$  a job in  $\tau_L$ ,  $t_d = d_{\ell,j}$ , and  $\mathcal{S}$  an EDF-hl schedule for  $\tau$  with the following property.

- (P) The tardiness of every job of every task  $T_k$  in  $\tau_L$  with deadline less than  $t_d$  is at most  $x + e_k$ , where  $x \geq 0$ .

Then, determining the smallest  $x$ , independent of the parameters of  $T_{\ell}$ , such that the tardiness of  $T_{\ell,j}$  remains at most  $x + e_{\ell}$  would by induction imply a tardiness of at most  $x + e_k$  for all jobs of tasks in  $\tau_L$ . Because  $\tau$  is arbitrary, the tardiness bound will hold for every concrete instantiation of  $\tau^N$ .

Our proof obligation is easily met if  $T_{\ell,j}$  completes by its deadline,  $t_d$ , so assume otherwise. The completion time of  $T_{\ell,j}$  depends on the amount of work that can compete with  $T_{\ell,j}$  after  $t_d$ . We follow the steps below to determine  $x$ .

- (S1) Compute an upper bound (UB) on the amount of work (including that due to  $T_{\ell,j}$ ) that can compete with  $T_{\ell,j}$  after  $t_d$ .

- (S2) Determine a lower bound (LB) on the amount of such work required for the tardiness of  $T_{\ell,j}$  to exceed  $x + e_{\ell}$ .

- (S3) Determine the smallest  $x$  such that the tardiness of  $T_{\ell,j}$  is at most  $x + e_{\ell}$  using UB and LB.

Let  $\Psi$  denote the set of all jobs with deadlines at most  $t_d$  of all tasks in  $\tau$ . Under EDF-hl, no job of a task in  $\tau_L$  with a deadline after  $t_d$  can compete with  $T_{\ell,j}$ . Therefore, competing work for  $T_{\ell,j}$  is given by (i) the amount of work pending at  $t_d$  for jobs in  $\Psi$ , i.e.,  $\text{LAG}(\Psi, t_d, \mathcal{S})$ , plus (ii) the amount of work demanded by jobs of tasks in  $\tau_H$  that are not in  $\Psi$  but can compete with jobs in  $\Psi$  in  $[t_d, t_d + x + e_{\ell})$ . We now determine an upper bound on these two components (step (S1) described above).

(In the analysis that follows, we assume that  $\Delta_h \ll x$  holds for all  $T_h$  in  $\tau_H$ . The analysis has to be extended slightly, otherwise. We have refrained from presenting a more general analysis in the interest of clarity.)

### 3.2.1 Upper Bound on $\text{LAG}(\Psi, t_d, \mathcal{S})$

Let the *carry-in* job of a task  $T_h$  in  $\tau_H$  be defined as that job of  $T_h$ , if any, with a release time before  $t_d$  and an absolute deadline afterward. Clearly, at most one such job exists for each  $T_h$ . Similarly, let the job of  $T_h$ , if any, with a release time before  $t_d + x + e_{\ell}$  and an effective deadline afterward be defined as its *carry-out* job. This is illustrated in Fig. 4. The carry-in job of  $T_h$  is its only job with an *absolute deadline* after  $t_d$  that may preempt (i.e., compete with) jobs in  $\Psi$  before  $t_d$  (i.e., become urgent before  $t_d$ ). Let  $\Psi_H$  be the set of all carry-in jobs of tasks in  $T_h$ . (For ease of reference, descriptions for these task sets and job sets are repeated in Fig. 2.)

By Lemma 2, the LAG of  $\Psi$  can increase only across a non-busy interval for  $\Psi$ . Recall that in a non-busy interval for  $\Psi$  fewer than  $m$  jobs from  $\Psi$  execute. In the case of an EDF-hl schedule, such a non-busy interval for  $\Psi$  can be classified into two types depending on whether a job from  $\Psi_H$  is executing in the interval while a ready job from  $\Psi$  is waiting. At the risk of slightly abusing terms,

$\tau_H$	$\stackrel{\text{def}}{=}$	Set of all privileged tasks in $\tau$
$\tau_L$	$\stackrel{\text{def}}{=}$	Set of all unprivileged tasks in $\tau$
$\Psi$	$\stackrel{\text{def}}{=}$	Set of all jobs of all tasks in $\tau$ with deadlines at most $t_d$
$\Psi_H$	$\stackrel{\text{def}}{=}$	Set of carry-in jobs of tasks in $\tau_H$

Figure 2: Task and job sets heavily referred to.

we will refer to the two types as *blocking* and *non-blocking non-busy* intervals. A *blocking non-busy interval* is one in which a job from  $\Psi_H$  is executing while a ready job from  $\Psi$  is waiting, whereas a *non-blocking, non-busy interval* is one in which fewer than  $m$  jobs from  $\Psi$  are executing, but there does not exist a ready job in  $\Psi$  that is waiting. Note that it is immaterial whether a job from  $\Psi_H$  is executing in a non-blocking, non-busy interval.

Before determining an upper bound on LAG, we state some needed properties. In [8], we showed that if a task does not execute continuously within a non-busy interval in an EDF schedule, then its lag at the end of the interval is at most zero. This property can be extended to a non-blocking, non-busy interval of an EDF-hl schedule, as follows.

**Lemma 3 (from [8])** *Let  $[t, t')$  be a maximally non-blocking, non-busy interval in  $[0, t_d)$  in  $\mathcal{S}$  and let  $T_k$  be a task in  $\tau$  with a job in  $\Psi$  that is active or pending at  $t'^-$ . If  $T_k$  does not execute continuously in  $[t, t')$ , then  $\text{lag}(T_k, t', \mathcal{S}) \leq 0$ .*

To see why this lemma holds, note that, because  $[t, t')$  is maximally non-busy and is non-blocking, at least one processor is idle throughout this interval, or a job from  $\Psi_H$  is executing while no job in  $\Psi$  is waiting. Recall that the absolute deadline of a job in  $\Psi_H$  is after  $t_d$ . Hence, if  $T_k$  is not executing at  $t'^-$ , then it has no pending work at  $t'$ , and hence, its lag at  $t'$  is at most zero. On the other hand, if  $T_k$  is executing at  $t'^-$ , but was not executing some time earlier in  $[t, t')$ , then it must have had no pending work when its most-recent job was released and must have executed continuously since then. In this case too, its lag cannot exceed zero.

The two lemmas that follow are proved in an appendix. The first lemma bounds the lag of a task in  $\tau_L$  at any arbitrary time at or before  $t_d$ . The second concerns the lags of tasks in  $\tau_H$ .

**Lemma 4** *Let  $v$  be an arbitrary time instant at or before  $t_d$ . Let  $T_k$  be a task in  $\tau_L$  and  $T_{k,q}$  its earliest pending job at  $v$ , and let  $\delta_{k,q} < e_k$  be the amount of time that  $T_{k,q}$  executed for before  $v$ . Then,  $\text{lag}(T_k, v, \mathcal{S}) \leq (v - d_{k,q}) \cdot u_k + e_k - \delta_{k,q}$ . Furthermore,  $v - d_{k,q} \leq x + \delta_{k,q}$ . Hence,  $\text{lag}(T_k, v, \mathcal{S}) \leq x \cdot u_k + e_k$ .*

**Lemma 5** *Let  $T_k$  be a task in  $\tau_H$  and  $T_{k,q}$  its earliest pending job at any arbitrary time  $v$ . Then,  $\text{lag}(T_k, v, \mathcal{S}) \leq \min(d_{k,q} + \Delta_k - v, e_k) + (v - d_{k,q}) \cdot u_k \leq e_k + \Delta_k \cdot u_k$ .*

We now turn to determining an upper bound on the LAG of  $\Psi$  at  $t_d$ . By Lemma 2, the LAG of  $\Psi$  can increase only across a non-busy interval for  $\Psi$ . Hence, an upper bound on LAG at the end of the latest non-busy interval before  $t_d$  across which LAG increases will serve as an upper bound for that at  $t_d$ . As discussed earlier, a non-busy interval in an EDF-hl schedule can be either blocking or non-blocking. We will consider these two cases separately. Let  $f$  be defined as follows.

$$f = \begin{cases} U_{\text{sum}}(\tau) - 1, & U_{\text{sum}}(\tau) \text{ is integral} \\ \lfloor U_{\text{sum}}(\tau) \rfloor, & \text{otherwise} \end{cases} \quad (9)$$

Expressions that occur frequently in the analysis are provided in Fig. 3. The lemma that follows shows how to bound LAG at the end of a non-blocking, non-busy interval.

**Lemma 6** *Let  $[t, t')$  be a maximally non-blocking, non-busy interval in  $[0, t_d)$  in  $\mathcal{S}$  and let  $\text{LAG}(\Psi, t', \mathcal{S}) > \text{LAG}(\Psi, t, \mathcal{S})$ . Then,  $\text{LAG}(\Psi, t', \mathcal{S}) \leq x \cdot U_L + U_H + E_L$ .*

**Proof:** By (7), the LAG of  $\Psi$  at  $t'$  is given by the sum of the lags at  $t'$  of all tasks in  $\tau$  with at least one job in  $\Psi$  that is active or pending at  $t'^-$ . By Lemma 3, the lag of such a task that does not execute continuously in  $[t, t')$  is at most zero. Hence,

$$\begin{aligned}
E_L &= \sum_{T_k \in \mathcal{E}_{\max}(\tau, f)} e_k \\
U_H &= \sum_{T_h \in \mathcal{U}_{\max}(\tau_H, \max(0, f-1-|\tau_L|))} \Delta_h \cdot u_h \\
E_H &= \sum_{T_h \in \tau_H} e_h(1-u_h) \\
U_L &= \sum_{T_k \in \mathcal{U}_{\max}(\tau_L, \min(f-1, |\tau_L|))} u_k \\
E'_H &= \sum_{T_h \in \tau_H} ((e_h(1-u_h) + u_h(e_\ell - \Delta_h) + \\
&\quad \min(e_h \cdot u_h, \Delta_h)) + \max(u_h(e_h - e_\ell), 0)) \\
U'_H &= \sum_{T_h \in \tau_H} u_h
\end{aligned}$$

Figure 3: Some expressions used in the paper.

to determine an upper bound on LAG at  $t'$ , it is sufficient to determine an upper bound on the lags of such tasks that are executing continuously in  $[t, t')$ . Let  $f'$  denote the number of such tasks. Then, by Lemma 2,

$$f' < \max_{t \leq \hat{t} < t'} \{U_{sum}(\Psi, \hat{t})\} \leq U_{sum}(\tau). \quad (10)$$

Let  $\alpha_H$  (resp.,  $\alpha_L$ ) denote the subset of all tasks in  $\tau_H$  (resp.,  $\tau_L$ ) that are executing continuously in  $[t, t')$  and have a job in  $\Psi$  that is active or pending at  $t'^-$ . Then,

$$|\alpha_H| + |\alpha_L| = f', \quad (11)$$

and by the above discussion on bounding LAG,

$$\begin{aligned}
&\text{LAG}(\Psi, t', \mathcal{S}) \\
&\leq \sum_{T_h \in \alpha_H} \text{lag}(T_h, t', \mathcal{S}) + \sum_{T_k \in \alpha_L} \text{lag}(T_k, t', \mathcal{S}) \\
&\leq \sum_{T_h \in \alpha_H} (\Delta_h \cdot u_h + e_h) + \sum_{T_k \in \alpha_L} (x \cdot u_k + e_k) \\
&\quad \text{\{by Lemmas 5 and 4\}} \\
&= \sum_{T_k \in \alpha_L \cup \alpha_H} e_k + \sum_{T_k \in \alpha_L} x \cdot u_k + \sum_{T_h \in \alpha_H} \Delta_h \cdot u_h \\
&\leq \sum_{T_k \in \mathcal{E}_{\max}(\tau, f')} e_k + \sum_{T_k \in \alpha_L} x \cdot u_k + \sum_{T_h \in \alpha_H} \Delta_h \cdot u_h \\
&\quad \text{\{By (11)\}} \\
&\leq \sum_{T_k \in \mathcal{E}_{\max}(\tau, f')} e_k + \sum_{T_k \in \mathcal{U}_{\max}(\tau_L, \min(f', |\tau_L|))} x \cdot u_k \\
&\quad + \sum_{T_h \in \mathcal{U}_{\max}(\tau_H, \max(0, f'-1-|\tau_L|))} \Delta_h \cdot u_h.
\end{aligned}$$

\{By (11) and assuming  $\Delta_h \ll x$  so that  $\Delta_h \cdot u_h < x \cdot u_k$  for all  $T_h \in \tau_H, T_k \in \tau_L\}$

Finally, as in [8], it can be shown that for LAG to increase across  $[t, t')$ , at least one job of  $\Psi$  with a deadline at or after  $t'$  should have completed execution before  $t$  and that at least one job executing at  $t$  should have a deadline at or after  $t'$ . Hence, the lag for its task  $T_k$  at  $t'$  is at most  $e_k$ . By this argument, the upper bound on LAG derived above reduces to

$$\begin{aligned}
&\text{LAG}(\Psi, t', \mathcal{S}) \\
&\leq \sum_{T_k \in \mathcal{E}_{\max}(\tau, f')} e_k + \sum_{T_k \in \mathcal{U}_{\max}(\tau_L, \min(f'-1, |\tau_L|))} x \cdot u_k \\
&\quad + \sum_{T_h \in \mathcal{U}_{\max}(\tau_H, \max(0, f'-1-|\tau_L|))} \Delta_h \cdot u_h.
\end{aligned}$$

The lemma follows because, by (9) and (10),  $f' \leq f$ .  $\blacksquare$

The next lemma shows how to bound LAG at the end of a blocking, non-busy interval.

**Lemma 7** *Let  $[t, t')$  be a blocking, non-busy interval in  $[0, t_d)$  in  $\mathcal{S}$  such that every instant in  $[t, t')$  is a blocking instant and any job of  $\Psi_H$  that executes in  $[t, t')$  executes continuously in  $[t, t')$ . Then,  $\text{LAG}(\Psi, t', \mathcal{S}) \leq \text{LAG}(\Psi, t, \mathcal{S}) + \sum_{T_h \in \alpha_H} (t' - t) \cdot (1 - u_h)$ , where  $\alpha_H$  is the subset of all tasks in  $\tau_H$  whose jobs in  $\Psi_H$  execute continuously in  $[t, t')$ .*

**Proof:** Let  $T_h$  be a task in  $\alpha_H$ , where  $\alpha_H$  is as defined in the statement of the lemma. Then, because the job of  $T_h$  that is executing in  $[t, t')$  is in  $\Psi_H$ ,  $T_h$  does not have a job in  $\Psi$  that is either active or pending anywhere in  $[t, t')$ . Thus, by (8),

$$(\forall \hat{t} : t \leq \hat{t} < t' :: U_{sum}(\Psi, \hat{t}) \leq U_{sum}(\tau) - \sum_{T_h \in \alpha_H} u_h), \quad (12)$$

and since the cumulative allocation at each instant in  $[t, t')$  in  $\text{PS}_\tau$  to jobs in  $\Psi$  is at most  $U_{sum}(\tau) - \sum_{T_h \in \alpha_H} u_h$ , the following holds.

$$A(\text{PS}_\tau, \Psi, t, t') \leq (t' - t) \left( U_{sum}(\tau) - \sum_{T_h \in \alpha_H} u_h \right) \quad (13)$$

Because  $[t, t')$  is continuously blocking, at every instant in  $[t, t')$ , there exists at least one job in  $\Psi$  that

is ready, but does not execute. This in turn implies that no processor is idle in the interval. Hence, we have the following.

$$A(\mathcal{S}, \Psi, t, t') = (t' - t)(m - |\alpha_H|) \quad (14)$$

By (13) and (14), and (6) (with  $t_1 = t$  and  $t_2 = t'$ ), we have

$$\begin{aligned} & \text{LAG}(\Psi, t', \mathcal{S}) \\ & \leq \text{LAG}(\Psi, t, \mathcal{S}) + \\ & \quad (t' - t) \left( (U_{\text{sum}}(\tau) - \sum_{T_h \in \alpha_H} u_h) - (m - |\alpha_H|) \right) \\ & = \text{LAG}(\Psi, t, \mathcal{S}) + \\ & \quad (t' - t) \left( (U_{\text{sum}}(\tau) - m) + |\alpha_H| - \sum_{T_h \in \alpha_H} u_h \right). \end{aligned}$$

Because  $U_{\text{sum}}(\tau) \leq m$ , the above implies  $\text{LAG}(\Psi, t', \mathcal{S}) \leq \text{LAG}(\Psi, t, \mathcal{S}) + (t' - t) \cdot \sum_{T_h \in \alpha_H} (1 - u_h)$ . ■

An upper bound on the LAG of  $\Psi$  at  $t_d$  can now be determined by combining Lemmas 6 and 7 as follows. (This lemma is proved in an appendix.)

**Lemma 8** *Let  $\delta_h \leq e_h$  denote the amount of time that the carry-in job (i.e., job in  $\Psi_H$ ), if any, of task  $T_h$  in  $\tau_H$  executes for before  $t_d$ . Then,  $\text{LAG}(\Psi, t_d, \mathcal{S}) \leq x \cdot U_L + U_H + E_L + \sum_{T_h \in \tau_H} \delta_h \cdot (1 - u_h)$ .*

To complete step (S1), we need to determine an upper bound on the work due to tasks in  $\tau_H$  that can compete with jobs in  $\Psi$  in  $[t_d, t_d + x + e_\ell]$ . We do this next.

### 3.2.2 Competing Demand by Jobs of Tasks in $\tau_H$ not in $\Psi$

Let  $D(T_h)$  denote the amount of work due to the jobs of a task  $T_h$  in  $\tau_H$  that are not in  $\Psi$  and that can compete with jobs of other tasks in  $\Psi$  in  $[t_d, t_d + x + e_\ell]$ . Then,  $D(T_h)$  is composed of three parts: (i) Work that needs to be done on a carry-in job, if any, (ii) mandatory work that needs to be done on a carry-out job, if any, and (iii) work to be done on all jobs that lie between the carry-in and carry-out

jobs, which is  $e_h$  times the number of such jobs. This is illustrated in Fig. 4. (Note that because the effective deadlines of any two consecutive jobs of  $T_h$  are separated by at least  $p_h$  time units, the latter of the two jobs does not become urgent until after the effective deadline of the former job has elapsed. Hence, no job released after the carry out job can compete with a job in  $\Psi$ .)

We now derive a bound on  $D(T_h)$ .

**Lemma 9** *Let  $T_h$  be any task in  $\tau_H$ . Then,  $D(T_h) \leq e_h - \delta_h + u_h \cdot (x + e_\ell - \Delta_h) + \min(0, \Delta_h - (e_h - \delta_h)u_h) + \max(0, u_h(e_h - e_\ell))$ , where  $\delta_h \leq e_h$  is the amount of time that the carry-in job, if any, of  $T_h$  executes before  $t_d$ .*

**Proof:** If no job of  $T_h$  has its effective deadline in  $[t_d, t_d + x + e_\ell]$ , then at most one job of  $T_h$  executes in the interval, and the maximum amount of time it executes for cannot exceed  $e_h - \delta_h$ . Therefore,  $D(T_h) \leq e_h - \delta_h$  holds. Assuming  $\Delta_h \ll x$ , it can be shown that  $u_h \cdot (x + e_\ell - \Delta_h) + \min(0, \Delta_h - (e_h - \delta_h)u_h) + \max(0, u_h(e_h - e_\ell)) \geq 0$  holds. Hence,  $D(T_h) \leq e_h - \delta_h + u_h \cdot (x + e_\ell - \Delta_h) + \min(0, \Delta_h - (e_h - \delta_h)u_h) + \max(0, u_h(e_h - e_\ell))$ , which proves the lemma.

Therefore, for the rest of the proof assume that at least one job of  $T_h$  has its effective deadline in  $[t_d, t_d + x + e_\ell]$ . Let  $T_{h,c_i}$  and  $T_{h,c_o}$  denote the carry-in and carry-out jobs of  $T_h$ , if any.

Let  $\xi_h = \rho_{h,c_i} - t_d$  and let  $\phi_h$  denote the offset from  $t_d + x + e_\ell$  of the last effective deadline in  $[t_d, t_d + x + e_\ell]$  of a job of  $T_h$ . Refer to Fig. 4. We now determine the three components of  $D(T_h)$  mentioned above.

**Work due to  $T_{h,c_i}$ .** Since  $T_{h,c_i}$  completes executing by  $\rho_{h,c_i}$ , the amount of time that  $T_{h,c_i}$  can execute for after  $t_d$  is at most  $\rho_{h,c_i} - t_d = \xi_h$  time units. Because  $T_{h,c_i}$  executes for  $\delta_h$  time units before  $t_d$ , it cannot execute for more than  $e_h - \delta_h$  time units after  $t_d$ . Thus, the amount of work to be done on  $T_{h,c_i}$  after  $t_d$  is at most  $\min(e_h - \delta_h, \xi_h)$ .

**Work due to  $T_{h,c_o}$ .** The effective deadline of  $T_{h,c_o}$  is separated from the previous effective deadline of  $T_h$  by at least  $p_h$  time units. Since the last

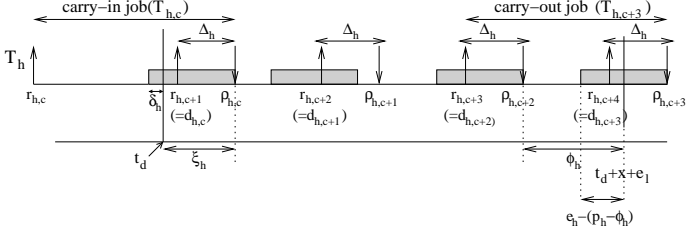


Figure 4: Competing demand due to task  $T_h$  in  $\tau_H$  in the interval  $[t_d, t_d + x + e_\ell]$ . Competing demand due to the carry-in job  $T_{h,c_i}$  ( $T_{h,c}$  here) is at most  $\min(e_h - \delta_h, \xi_h)$  and that due to the carry-out job  $T_{h,c_o}$  ( $T_{h,c+3}$  here) is at most  $\max(0, e_h - (p_h - \phi_h))$ .

effective deadline within  $[t_d, t_d + x + e_\ell]$  is  $\phi_h$  time units before  $t_d + x + e_\ell$ ,  $\rho_{h,c_o}$  is at least  $p_h - \phi_h$  time units after  $t_d$ . Therefore,  $\min(e_h, p_h - \phi_h)$  units of work due to  $T_{h,c_o}$  does not compete with jobs in  $\Psi$  before  $t_d + x + e_\ell$ . Hence, the competing work in  $[t_d, t_d + x + e_\ell]$  due to the carry-out job is at most  $\max(0, e_h - (p_h - \phi_h))$ .

**Work due to jobs between  $T_{h,c_i}$  and  $T_{h,c_o}$ .** The effective deadlines of successive jobs of  $T_h$  are separated by at least  $p_h$  time units. Therefore, the number of jobs of  $T_h$  that lie between  $\rho_{h,c_i}$  and  $t_d + x + e_\ell - \phi_h$  is at most  $\lfloor \frac{x+e_\ell-\xi_h-\phi_h}{p_h} \rfloor \leq \frac{x+e_\ell-\xi_h-\phi_h}{p_h}$ . Combining the three components above, we have

$$\begin{aligned}
D(T_h) &\leq \left( \frac{x+e_\ell-\xi_h-\phi_h}{p_h} \right) \cdot e_h + \\
&\quad \max(0, e_h - (p_h - \phi_h)) + \min(e_h - \delta_h, \xi_h) \\
&= \max((x+e_\ell-\xi_h-\phi_h)u_h, (x+e_\ell-\xi_h-\phi_h)u_h \\
&\quad + e_h - (p_h - \phi_h)) + \min(e_h - \delta_h, \xi_h) \\
&\quad \{\text{Because } e_h/p_h = u_h\} \\
&\leq u_h(x+e_\ell-\xi_h) + \min(e_h - \delta_h, \xi_h) \\
&\quad \{\text{Because } 0 \leq \phi_h \leq p_h \text{ and } u_h \leq 1\} \\
&= u_h(x+e_\ell-\Delta_h-\chi_h) + \min(e_h - \delta_h, \Delta_h + \chi_h) \\
&\quad \{\text{Letting } \xi_h = \Delta_h + \chi_h; \text{ because } d_{h,c_i} > t_d, \chi_h > 0\} \\
&= \min(e_h - \delta_h + u_h(x+e_\ell-\Delta_h-\chi_h), \\
&\quad \chi_h(1-u_h) + \Delta_h + u_h(x+e_\ell-\Delta_h)) \\
&= \min(e_h - \delta_h + u_h(x+e_\ell-\Delta_h-\chi_h), \\
&\quad (e_h - \delta_h)(1-u_h) + \Delta_h + u_h(x+e_\ell-\Delta_h)) \\
&\quad \{\text{Because } \chi_h + \Delta_h < e_h - \delta_h \Rightarrow \chi_h < e_h - \delta_h\} \\
&\leq \min(e_h - \delta_h + u_h(x+e_\ell-\Delta_h),
\end{aligned}$$

$$\begin{aligned}
&(e_h - \delta_h)(1-u_h) + \Delta_h + u_h(x+e_\ell-\Delta_h) \\
&\quad \{\text{Because } \chi_h > 0\} \\
&\leq e_h - \delta_h + u_h(x+e_\ell-\Delta_h) + \\
&\quad \min(0, \Delta_h - (e_h - \delta_h)u_h). \\
&\leq e_h - \delta_h + u_h(x+e_\ell-\Delta_h) + \\
&\quad \min(0, \Delta_h - (e_h - \delta_h)u_h) + \max(0, u_h(e_h - e_\ell)).
\end{aligned}$$

■

The next lemma gives a bound on the sum of the LAG of  $\Psi$  and the competing work due to tasks in  $\tau_H$ .

**Lemma 10**  $\text{LAG}(\Psi, t_d, \mathcal{S}) + \sum_{T_h \in \tau_H} D(T_h) \leq L + \sum_{T_h \in \tau_H} (e_h(1-u_h) + u_h \cdot (x+e_\ell-\Delta_h) + \min(e_h \cdot u_h, \Delta_h) + \max(0, u_h(e_h - e_\ell)))$ , where  $L = x \cdot U_L + U_H + E_L$ .

**Proof:** By Lemma 8,

$$\text{LAG}(\Psi, t_d, \mathcal{S}) \leq L + \sum_{T_h \in \tau_H} \delta_h \cdot (1-u_h), \quad (15)$$

where  $L = x \cdot U_L + U_H + E_L$  and  $\delta_h$  is the amount of time the carry-in job of  $T_h$  in  $\Psi_H$  executed before  $t_d$ . By Lemma 9,

$$\begin{aligned}
&\sum_{T_h \in \tau_H} D(T_h) \\
&\leq \sum_{T_h \in \tau_H} (e_h - \delta_h + u_h \cdot (x+e_\ell-\Delta_h) \\
&\quad + \min(0, \Delta_h - (e_h - \delta_h)u_h) + \max(0, u_h(e_h - e_\ell))) \\
&\quad (16)
\end{aligned}$$

By (15) and (16), we have

$$\begin{aligned}
&\text{LAG}(\Psi, t_d, \mathcal{S}) + \sum_{T_h \in \tau_H} D(T_h) \\
&\leq L + \sum_{T_h \in \alpha} (\delta_h(1-u_h) + e_h - \delta_h + u_h(x+e_\ell-\Delta_h) \\
&\quad + \min(0, \Delta_h - (e_h - \delta_h)u_h) + \max(0, u_h(e_h - e_\ell))) \\
&= L + \sum_{T_h \in \tau_H} ((e_h - \delta_h \cdot u_h) \\
&\quad + u_h(x+e_\ell-\Delta_h) - (e_h - \delta_h)u_h \\
&\quad + \min((e_h - \delta_h)u_h, \Delta_h) + \max(0, u_h(e_h - e_\ell))) \\
&\leq L + \sum_{T_h \in \tau_H} (e_h(1-u_h) + u_h(x+e_\ell-\Delta_h) \\
&\quad + \min(e_h \cdot u_h, \Delta_h) + \max(0, u_h(e_h - e_\ell))).
\end{aligned}$$

That completes step (S1). The next step is to determine a lower bound on the amount of such work required for tardiness of  $T_{\ell,j}$  to exceed a certain amount, which we do next.

### 3.3 Lower Bound on LAG + D

**Lemma 11** *If  $\text{LAG}(\Psi, t_d, \mathcal{S}) \leq (m - |\tau_H|) \cdot x + e_\ell$  and  $|\tau_H| < m$  or  $\text{LAG}(\Psi, t_d, \mathcal{S}) + \sum_{T_h \in \tau_H} \mathbf{D}(T_h) \leq (m - \max(|\tau_H| - 1, 0) \cdot u_\ell) \cdot x + e_\ell$  and  $|\tau_H| > 0$ , then the tardiness of  $T_{\ell,j}$  in  $\mathcal{S}$  is at most  $x + e_\ell$ .*

**Proof:** To prove the lemma, we show that  $T_{\ell,j}$  completes executing by  $t_d + x + e_\ell$ . If  $j > 1$ , then  $d_{\ell,j-1} \leq t_d - p_\ell$  holds, and by (P), we have the following.

(R)  $T_{\ell,j-1}$  completes executing by  $t_d + x + e_\ell - p_\ell$ , for  $j > 1$ .

We consider the two conditions stated in the lemma in two separate cases below. In what follows, let  $H = |\tau_H|$ .

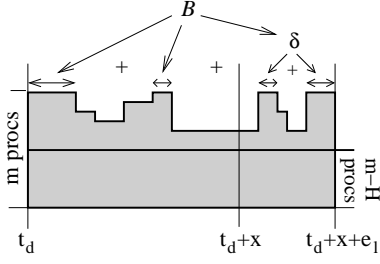
**Case 1:**  $\text{LAG}(\Psi, t_d, \mathcal{S}) \leq (m - |\tau_H|) \cdot x + e_\ell$  and  $|\tau_H| < m$ . Let  $\delta_{\ell,j}$  denote the amount of time that  $T_{\ell,j}$  executed before  $t_d$ . By the conditions of this case, the amount of work pending at  $t_d$  for jobs in  $\Psi$ , and hence for those of  $\tau_L$  in  $\Psi$  is at most  $(m - H) \cdot x + e_\ell$ . Without loss of generality, assume that the jobs in  $\Psi$  are the only jobs of  $\tau_L$  (or, equivalently, jobs with deadlines beyond  $t_d$  have been discarded). Hence, if  $m - H$  tasks of  $\tau_L$  are executing at any instant in  $[t_d, t_d + x + \frac{\delta_{\ell,j}}{m-H})$ , then the amount of work done in the interval on jobs of  $\tau_L$  in  $\Psi$  is at least  $(m - H)(x + \frac{\delta_{\ell,j}}{m-H})$ . Therefore, the amount of work pending at  $t_d + x + \frac{\delta_{\ell,j}}{m-H}$  for those jobs is at most  $(m - H) \cdot x + e_\ell - (m - H)(x + \frac{\delta_{\ell,j}}{m-H}) = e_\ell - \delta_{\ell,j}$ . Since  $T_{\ell,j}$  has executed for  $\delta_{\ell,j}$  time before  $t_d$ , the latest time that  $T_{\ell,j}$  completes executing is  $t_d + x + \frac{\delta_{\ell,j}}{m-H} + e_\ell - \delta_{\ell,j} < t_d + x + e_\ell$ . If fewer than  $m - H$  tasks are executing at some time, say  $t' < t_d + x + \frac{\delta_{\ell,j}}{m-H}$ , then fewer than  $m - H$

■ tasks of  $\tau_L$  have pending work at  $t'$ . Because tasks of  $\tau_H$  can execute on at most  $H$  processors at any instant,  $T_\ell$  can execute uninterruptedly from  $t'$  until  $T_{\ell,j}$  completes execution. Suppose the job of  $T_\ell$  executing at  $t'$  is  $T_{\ell,j}$ . Then, since  $t' < t_d + x + \frac{\delta_{\ell,j}}{m-H}$  holds, and the amount of work pending for  $T_{\ell,j}$  is at most  $e_\ell - \delta_{\ell,j}$ ,  $T_{\ell,j}$  completes executing before  $t_d + x + e_\ell$ . So, assume that a prior job of  $T_\ell$  is executing at  $t'$ . In this case,  $T_{\ell,j}$  could not have executed before  $t_d$ , and hence,  $\delta_{\ell,j} = 0$ , which implies (from the definition of  $t'$ ) that  $t' < t_d + x$ . Furthermore,  $j \geq 2$  holds, and by (R),  $T_{\ell,j-1}$  completes executing by  $t_d + x$ , and hence, the latest time that  $T_{\ell,j}$  commences execution is at or before  $t_d + x$ , and so the latest time that  $T_{\ell,j}$  completes execution is  $t_d + x + e_\ell$ .

**Case 2:**  $\text{LAG}(\Psi, t_d, \mathcal{S}) + \sum_{T_h \in \tau_H} \mathbf{D}(T_h) \leq (m - |\tau_H|) \cdot u_\ell \cdot x + e_\ell$ . At the risk of some notational abuse, let a time interval (resp., instant) in  $[t_d, t_d + x + e_\ell)$  in which all  $m$  processors are executing a job of  $\Psi$  or that part of a task in  $\tau_H$  that can compete with  $\Psi$  be referred to as a *busy interval* (resp., *instant*). Then, if pending, task  $T_\ell$  can execute in every non-busy instant. If the latest busy instant in  $[t_d, t_d + x + e_\ell)$  is at or before  $t_d + x$ , then because, by (R),  $T_{\ell,j-1}$ , if it exists, completes execution at or before  $t_d + x$ , the latest time that  $T_{\ell,j}$  completes execution is  $t_d + x + e_\ell$ .

So, for the rest of this proof we assume that the latest busy instant is after  $t_d + x$ . Let the total lengths of the busy intervals in  $[t_d + x, t_d + x + e_\ell)$  be  $\delta \leq e_\ell$ . (Refer to Fig. 5.) Therefore,  $T_\ell$  can execute for at least  $e_\ell - \delta$  time in that interval. If fewer than  $m - H + 1$  tasks are executing at any non-busy instant  $t_n$  at or before  $t_d + x$ , then at most  $m - H$  tasks of  $\tau_L$  have pending work at or after  $t_n$ . Hence, since tasks in  $\tau_H$  can execute on at most  $H$  processors at any instant,  $T_\ell$  is guaranteed uninterrupted execution from  $t_n$  until  $T_{\ell,j}$  completes. Hence, since by (R),  $T_{\ell,j-1}$  (if it exists) completes execution by  $t_d + x$ ,  $T_{\ell,j}$  would complete execution no later than  $t_d + x + e_\ell$ . Therefore, for the rest of this case, assume the following.

(N) At least  $\min(m - H + 1, m)$  tasks are executing



$B$  : total length of all the busy intervals in  $[t_d, t_d + x + e_l)$   
 $\delta$  : total length of all the busy intervals in  $[t_d + x, t_d + x + e_l)$

Figure 5: Case 2 of Lemma 11. Sample schedule in  $[t_d, t_d + x + e_l)$ .

at every non-busy instant in  $[t_d, t_d + x)$ .

Let  $B$  denote the total length of all the busy intervals in  $[t_d, t_d + x + e_l)$ . (Refer to Fig. 5.) If  $B \leq x - x \cdot u_\ell$ , then  $T_\ell$  can execute for at least  $x \cdot u_\ell + e_\ell$  time in  $[t_d, t_d + x + e_l)$ . By Lemma 4,  $\text{lag}(T_\ell, t_d, \mathcal{S}) \leq x \cdot u_\ell + e_\ell$ , and hence,  $T_{\ell,j}$  would complete executing at or before  $t_d + x + e_\ell$ . So, assume  $B = x - x \cdot u_\ell + \delta_1$ , where  $\delta_1 > 0$ . With this assumption, we now compute the total amount of work done in  $[t_d, t_d + x + e_l)$ . The total amount of work done in all busy intervals in  $[t_d, t_d + x + e_l)$  is  $m \cdot B$ . By (N), at least  $\min(m - H + 1, m)$  tasks are executing at every non-busy instant in  $[t_d, t_d + x)$ . The total length of all non-busy intervals in  $[t_d, t_d + x)$  is  $x - (B - \delta)$ . Therefore, the amount of work done in all non-busy intervals in  $[t_d, t_d + x)$  is at least  $\min(m - H + 1, m) \cdot (x - B + \delta)$ . The total length of all non-busy intervals in  $[t_d + x, t_d + x + e_l)$  is  $e_\ell - \delta$ , and at least task  $T_\ell$  of  $\tau_L$  has pending jobs in  $\Psi$  until  $t_d + x + e_\ell$ , and hence, executes in every non-busy instant in  $[t_d + x, t_d + x + e_l)$ . (Otherwise, it would imply that  $T_{\ell,j}$  has completed executing before  $t_d + x + e_\ell$ , completing the proof, as well). Hence, the total amount of work done in  $[t_d, t_d + x + e_l)$  is at least  $mB + \min(m - H + 1, m) \cdot (x - B + \delta) + (e_\ell - \delta)$ , which, on substituting  $x - x \cdot u_\ell + \delta_1$  for  $B$ , simplifies to  $m x - H \cdot x \cdot u_\ell + (m - H) \cdot \delta + x \cdot u_\ell + H \cdot \delta_1 + (e_\ell - \delta)$  for  $H > 0$  and  $m \cdot (x + \delta) + e_\ell - \delta$  for  $H = 0$ .

By the condition of this case, the amount of work that needs to be done in  $[t_d, t_d + x + e_l)$  for jobs in  $\Psi$  and of tasks in  $\tau_H$  that can compete with  $\Psi$  is at most  $m x - \max(H - 1, 0) \cdot x \cdot u_\ell + e_\ell$ . Therefore, the amount of work pending at  $t_d + x + e_\ell$  is at most

$-(H - 1) \cdot \delta_1 - (m - H) \cdot \delta$ , for  $1 \leq H \leq m$ , and is at most  $-(m - 1) \cdot \delta$ , for  $H = 0$ . Because  $\delta$  and  $\delta_1$  are positive, both the above bounds are negative. Thus, no work of jobs in  $\Psi$ , and in particular, that of  $T_{\ell,j}$ , can be pending at  $t_d + x + e_\ell$ . ■

This completes step (S2). We are left with determining a value for  $x$  for which the tardiness of  $T_{\ell,j}$  is at most  $x + e_\ell$ .

### 3.4 Finishing Up

Solving for  $x$  using Lemma 8 and the first condition in Lemma 11, *i.e.*, solving for  $x$  in  $x \cdot U_L + U_H + E_L + E_H \leq (m - |\tau_H|)x + e_\ell$ , yields

$$x \geq \frac{E_L + U_H + E_H - e_\ell}{(m - |\tau_H|) - U_L}, \quad (17)$$

where  $E_H$  is as in Fig. 3. Solving using Lemma 10 and the second condition of Lemma 11, *i.e.*, using  $\sum_{T_h \in \tau_H} (e_h \cdot (1 - u_h) + u_h \cdot (x + e_\ell - \Delta_h) + \min(e_h \cdot u_h, \Delta_h) + \max(0, u_h \cdot (e_h - e_\ell))) + x \cdot U_L + U_H + E_L \leq m x - \max(|\tau_H| - 1, 0) \cdot x \cdot u_\ell + e_\ell$ , we have

$$x \geq \frac{E_L + U_H + E'_H - e_\ell}{m - \max(|\tau_H| - 1, 0) \cdot u_\ell - U_L - U'_H}, \quad (18)$$

where  $E'_H$  and  $U'_H$  are as in Fig. 3. Hence, if  $x$  is smaller of the two values that are on the right-hand sides of (17) and (18), then the tardiness of  $T_{\ell,j}$  would not exceed  $x + e_\ell$ . A value of  $x$  that is independent of the parameters of  $T_\ell$  is obtained by replacing  $e_\ell$  by  $e_{\min}$  and  $u_\ell$  by  $u_{\max}(\tau_L)$  in (17) and (18). Similarly, the  $e_\ell$  term in the expression for  $E'_H$  has to be replaced by  $e_{\max}(\tau_L)$ . By inducting on the jobs of  $\tau_L$  in the non-decreasing order of their deadlines, we have the following theorem.

**Theorem 1** EDF-hl ensures a tardiness of at most  $\min(X_1, X_2) + e_k$  to every task  $T_k$  of  $\tau_L$  if  $|\tau_H| \leq m$  and  $U_{\text{sum}}(\tau) \leq m$ , where  $X_1 = \frac{E_L + U_H + E_H - e_{\min}(\tau_L)}{(m - |\tau_H|) - U_L}$  and  $X_2 = \frac{E_L + U_H + E'_H - e_{\min}(\tau_L)}{m - \max(|\tau_H| - 1, 0) \cdot u_{\max}(\tau_L) - U_L - U'_H}$ .

**Conditions for bounded tardiness.** Since the derivation was based on the assumption that  $x \geq 0$ ,  $X_1$  and  $X_2$  are valid only if their denominators are

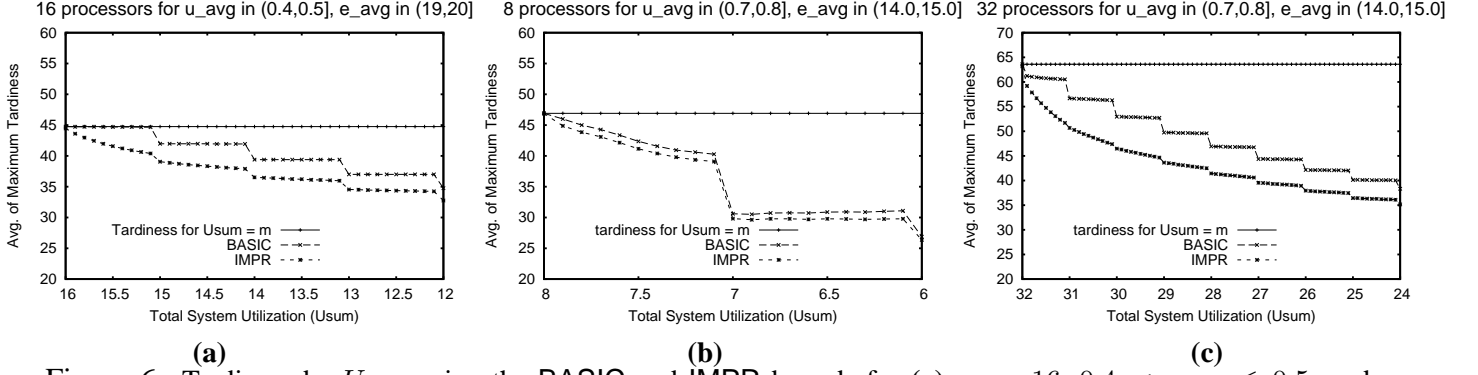


Figure 6: Tardiness by  $U_{sum}$  using the BASIC and IMPR bounds for (a)  $m = 16$ ,  $0.4 < u_{avg} \leq 0.5$ , and  $19.0 < e_{avg} \leq 20.0$ , (b)  $m = 8$ ,  $0.7 < u_{avg} \leq 0.8$ , and  $14.0 < e_{avg} \leq 15.0$ , and (c)  $m = 32$ ,  $0.7 < u_{avg} \leq 0.8$ , and  $14.0 < e_{avg} \leq 15.0$ .

non-negative.  $X_1$  and  $X_2$  are bounded only if their denominators are greater than zero. Hence, if the sum of the utilizations of the  $f - 1$  heaviest tasks in  $\tau_L$  is less than  $m - |\tau_H|$ , then  $X_1$  is bounded. Similarly,  $X_2$  is bounded only if the sum of the utilizations of the heaviest  $f - 1$  tasks in  $\tau_L$  is less than  $m - |\tau_H| \cdot u_{\max}(\tau_L) - U'_H$ . Hence, if either of the above conditions holds, then bounded tardiness is guaranteed to tasks in  $\tau_L$ .

**Computational complexity.** Each of the  $E$  and  $U$  terms in the tardiness bound can be computed in  $O(f)$  time. The  $f$  tasks with highest utilizations or execution costs can be selected from  $\tau$ ,  $\tau_L$ , or  $\tau_H$  in  $O(n)$  time. Hence, the tardiness bound can be computed in  $O(n)$  time.

## 4 Experimental Evaluation

In this section, we present the results of experiments conducted to (i) determine the range of the tardiness bound guaranteed by EDF-hl on an average and (ii) evaluate the tardiness-utilization trade-off possible in the absence of high-priority tasks. Due to space constraints, only a subset of the results is presented here.

**Tardiness-Utilization trade-off.** As mentioned earlier, EDF-hl reduces to EDF in the absence of high-priority tasks. Hence, in this case, the tardiness bound given in Thm. 1 applies to every task in  $\tau$ . Note that the tardiness bound is expressed in

terms of  $U_{sum}(\tau)$  in addition to individual task parameters. Hence, an alternative to EDF-hl for guaranteeing lower tardiness is to lower  $U_{sum}$ . This approach may be preferable if a majority of the tasks require lower tardiness and the gains are reasonable for slight decreases in  $U_{sum}$ .

In the absence of high priority tasks, using a slightly different, but more complicated, analysis than that used in Sec. 3 or in [8], it can be shown that

$$U_L \leq \sum_{T_k \in U_{\max}(\tau_L, f-1)} \frac{u_k^2(m-f)}{(m-U_{sum}) + u_k(U_{sum}-f)}, \quad (19)$$

which when used in the expression for the tardiness bound in Thm. 1 results in slightly lower values. We will refer to the bound given in Thm. 1 as BASIC and the bound obtained by using (19) as IMPR.

We evaluated the tardiness-utilization trade-off that is possible by generating random task sets with varying values for  $U_{sum}$  and computing the BASIC and IMPR bounds for each and comparing these bounds with those obtained from our earlier work, when  $U_{sum} = m$  [8]. Simulation experiments were conducted for four, eight, 16, and 32 processors, with  $U_{sum}$  varying between  $3m/4$  and  $m$  in increments of 0.1. 600,000 task sets, with at least  $m + 1$  tasks in each, were generated for each  $(U_{sum}, m)$  pair. The maximum utilization of any task in a task set varied uniformly from 0.5 to 1.0. The task sets generated were grouped based on  $u_{avg}$  and  $e_{avg}$ , where  $u_{avg}$  and  $e_{avg}$  are the av-

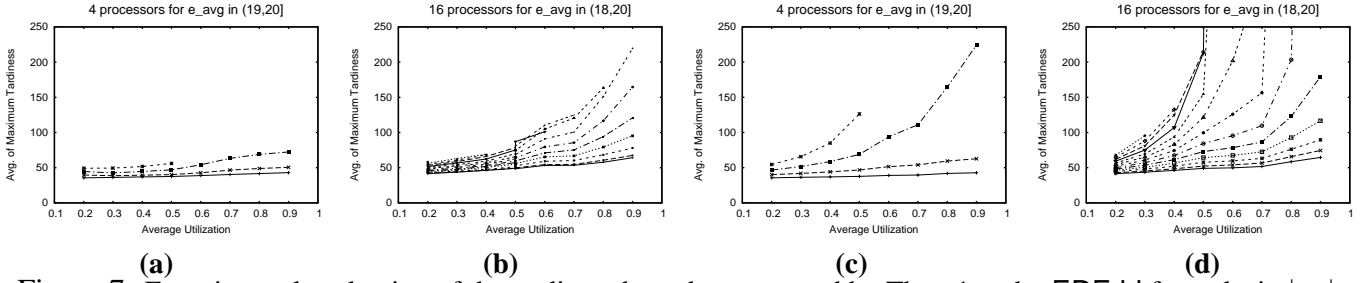


Figure 7: Experimental evaluation of the tardiness bounds guaranteed by Thm. 1 under EDF-hl for tasks in  $|\tau_L|$ . LU with (a)  $m = 4$  and (b)  $m = 16$ . HU with (c)  $m = 4$  and (d)  $m = 16$ . The different curves in each inset correspond to different values of  $|\tau_H|$ .  $|\tau_H| = 0$  for the bottom-most curve and is greater by one for each curve higher up.

erages of the highest  $\lfloor U_{sum} \rfloor$  task utilizations and execution costs, respectively. The variation in tardiness (mean of the maximum tardiness for all task sets in a group) with  $U_{sum}$  for (i)  $m = 16$  when  $0.4 < u_{avg} \leq 0.5$  and  $19.0 < e_{avg} \leq 20.0$  and (ii)  $m = 8$  and  $m = 32$  when  $0.7 < u_{avg} \leq 0.8$  and  $14.0 < e_{avg} \leq 15.0$  are presented in Fig. 6. Note that the rate at which tardiness drops with decreasing  $U_{sum}$  is higher when  $u_{avg}$  is higher (in the  $(0.7, 0.8]$  range). Furthermore, the rate at which tardiness drops with  $U_{sum}$  decreases with decreasing  $U_{sum}$ . For instance, in inset (c), reducing  $U_{sum}$  to 31.0 (which is 96.8% of  $m (= 32)$ ) lowers tardiness to less than 50.0 from over 60.0, which is a drop of over 20%, whereas to lower tardiness to less than 40.0,  $U_{sum}$  has to be decreased to approximately 27.0 (which is 84.3% of  $m$ ). Hence, setting  $U_{sum}$  to a value slightly lower than  $m$  may be appropriate when high utilization tasks are present in the task system. At this point, we would like to note that these characteristics should be attributed to the bounds derived (and to the analysis) and not to the algorithm per se.

**Tardiness bounds for EDF-hl.** We also experimentally evaluated the tardiness bounds that can be guaranteed to low-priority tasks on an average under EDF-hl for  $m = 4$  and  $m = 16$ , with  $U_{sum} = m$ . The task sets generated were grouped based on the average of the  $m$  highest task utilizations and the utilizations of the tasks in  $\tau_H$ , denoted  $u_{avg}$ . ( $e_{avg}$  is with respect to execution costs, analogously.) For each task set generated, the number of tasks in  $\tau_H$  was varied from zero to  $m$ , and for

	$u_{avg}$						
$ \tau_H $	0.2	0.3	0.4	0.5	0.6	0.7	0.8
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	1.9	100.0
8	0.0	0.0	0.0	0.0	37.2	98.09	100.0
9	0.0	0.0	0.0	0.0	99.3	100.0	100.0
10	0.0	0.0	0.0	0.0	100.0	100.0	100.0
11	0.0	0.0	0.0	14.76	100.0	100.0	100.0
12	0.0	0.0	25.0	100.0	100.0	100.0	100.0
13	0.0	9.7	99.78	100.0	100.0	100.0	100.0
14	0.0	99.67	100.0	100.0	100.0	100.0	100.0
15	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Table 1: Percentage of task sets for which unbounded tardiness was computed for  $m = 16$  under HU.

each  $|\tau_H|$ , the members of  $\tau_H$  were chosen in two different ways: first, as tasks with the highest  $|\tau_H|$  utilizations in the generated task set (denoted HU), and then, as tasks with the lowest  $|\tau_H|$  utilizations (denoted LU). The variation in tardiness with  $u_{avg}$  as the number of high-priority tasks is increased is plotted in Fig. 7 for both HU and LU. As expected, tardiness increases with  $|\tau_H|$  and  $u_{avg}$ , and the increase is higher for HU than for LU. The tardiness bounds computed grew to unbounded values for certain task sets at high values of  $|\tau_H|$ , with the percentage of such task sets increasing with increasing  $u_{avg}$ . The percentage of such task sets for HU is tabulated by  $|\tau_H|$  and  $u_{avg}$  for  $m = 16$  in Table 1. The figures for LU are slightly lower.

## 5 Conclusion

We have addressed the issue of supporting tasks whose tolerance to tardiness is lower than that currently known to be possible under EDF. We have proposed a new scheduling policy called EDF-hl, which is based on EDF, and have shown that under EDF-hl, a limited number of *privileged* tasks can be guaranteed any tardiness, including zero tardiness, and that bounded tardiness can be guaranteed to the remaining tasks if their utilizations are restricted. The tardiness bound derived is a function of  $U_{sum}$ , in addition to individual task parameters, and hence, tardiness for all tasks can be lowered by slightly lowering  $U_{sum}$ . We have, through simulations, assessed the impact of privileged tasks on the tardiness bounds that can be guaranteed to the remaining tasks, and the tardiness-utilization trade-off that is possible in the absence of privileged tasks.

This problem of supporting sporadic tasks with different tardiness requirements may alternatively be viewed as one of supporting tasks with relative deadlines at least periods. The EDF schedulability tests available for task systems with relative deadlines equal to periods on a multiprocessor, though applicable when deadlines may exceed periods also, are pessimistic and tend to under-utilize the underlying platform. The work presented in this paper is an attempt towards remedying this limitation.

## References

- [1] J. Anderson, V. Bud, and U. Devi. An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In *Proc. of the 17th Euromicro Conference on Real-time Systems*, pages 199–208, July 2005.
- [2] B. Andersson and J. Jonsson. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In *Proc. of the 15th Euromicro Conference on Real-time Systems*, pages 33–40, July 2003.
- [3] S. Baruah. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Transactions on Computers*, 53(6):781–784, 2004.
- [4] S. Baruah, N. Cohen, C.G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.
- [5] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In Joseph Y. Leung, editor, *Handbook on Scheduling Algorithms, Methods, and Models*, pages 30.1–30.19. Chapman Hall/CRC, Boca Raton, Florida, 2004.
- [6] S. Cho, S.K. Lee, S. Ahn, and K.J. Lin. Efficient real-time scheduling algorithms for multiprocessors. *IEICE Transactions on Communications*, E85-B(12):2859–2867, December 2002.
- [7] M. Dertouzos. Control robotics: The procedural control of physical processes. In *Proc. of IFIP Cong.*, pages 807–813, 1974.
- [8] U. Devi and J. Anderson. Tardiness bounds for global EDF scheduling on a multiprocessor. In *Proc. of the 26th IEEE Real-time Systems Symposium*, Dec. 2005. To appear.
- [9] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.
- [10] J.M. Lopez, M. Garcia, J.L. Diaz, and D.F. Garcia. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In *Proc. of the 12th Euromicro Conference on Real-time Systems*, pages 25–34, June 2000.

- [11] A. Srinivasan and J. Anderson. Efficient scheduling of soft real-time applications on multiprocessors. In *Proc. of the 15th Euromicro Conference on Real-time Systems*, pages 51–59, July 2003.
- [12] P. Valente and G. Lipari. An upper bound to the lateness of soft real-time tasks scheduled by EDF on multiprocessors. In *Proc. of the 26th IEEE Real-time Systems Symposium*, Dec. 2005. To appear.

## Appendix: Additional Proofs

**Lemma 4** *Let  $v$  be an arbitrary time instant at or before  $t_d$ . Let  $T_k$  be a task in  $\tau_L$  and  $T_{k,q}$  its earliest pending job at  $v$ , and let  $\delta_{k,q} < e_k$  be the amount of time that  $T_{k,q}$  executed for before  $v$ . Then,  $\text{lag}(T_k, v, \mathcal{S}) \leq (v - d_{k,q}) \cdot u_k + e_k - \delta_{k,q}$ . Furthermore,  $v - d_{k,q} \leq x + \delta_{k,q}$ . Hence,  $\text{lag}(T_k, v, \mathcal{S}) \leq x \cdot u_k + e_k$ .*

**Proof:** We prove the lemma for the case  $d_{k,q} \leq v$ , leaving the case  $d_{k,q} > v$  to the reader. The amount of work pending for  $T_{k,q}$  at  $v$  is  $e_k - \delta_{k,q}$ .  $T_k$  is allocated at most  $u_k$  time at every instant after  $d_{k,q}$  in  $\text{PS}_\tau$ . The first lag bound indicated follows from these two facts. The bound on  $v - d_{k,q}$  follows from (P) and the second lag bound is obtained by substituting the bound for  $v - d_{k,q}$  in the first lag bound. ■

**Lemma 5** *Let  $T_k$  be a task in  $\tau_H$  and  $T_{k,q}$  its earliest pending job at any arbitrary time  $v$ . Then,  $\text{lag}(T_k, v, \mathcal{S}) \leq \min(d_{k,q} + \Delta_k - v, e_k) + (v - d_{k,q}) \cdot u_k \leq e_k + \Delta_k \cdot u_k$ .*

**Proof:** Because  $T_k$  is in  $\tau_H$  and  $T_{k,q}$  is pending at  $v$ ,  $v \leq d_{k,q} + \Delta_k$  holds. Since  $T_{k,q}$  is guaranteed continuous execution until completion after time  $d_{k,q} + \Delta_k - e_k$ , the amount of work pending for  $T_{k,q}$  at  $v$  is at most  $\min(d_{k,q} + \Delta_k - v, e_k)$ , i.e.,  $A(\mathcal{S}, T_{k,q}, r_{k,q}, v) \geq e_k - \min(d_{k,q} + \Delta_k - v, e_k)$ . If  $v \leq d_{k,q}$ , then  $A(\text{PS}_\tau, T_{k,q}, r_{k,q}, v) = e_k - (d_{k,q} - v) \cdot u_k$ . Thus,  $\text{lag}(T_{k,q}, v, \mathcal{S}) = A(\text{PS}_\tau, T_{k,q}, r_{k,q}, v) - A(\mathcal{S}, T_{k,q}, r_{k,q}, v) \leq \min(d_{k,q} + \Delta_k - v, e_k) - (d_{k,q} - v) \cdot u_k$ . Because  $T_{k,q}$  is the earliest pend-

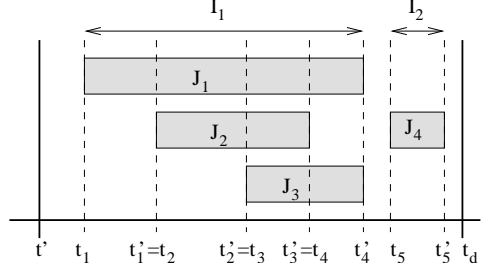


Figure 8: Lemma 8. There does not exist a non-blocking, non-busy interval across which LAG increases in  $[t', t_d]$ .  $J_1, \dots, J_4$  are jobs in  $\Psi_H$ . Their execution in blocking, non-busy intervals  $I_1$  and  $I_2$  is shown, as well as the slicing of the blocking interval  $I_1$  as specified in the proof.

ing job of  $T_k$  at  $v$  and no later job of  $T_k$  can be released before  $d_{k,q}$ ,  $\text{lag}(T_k, v, \mathcal{S}) = \text{lag}(T_{k,q}, v, \mathcal{S}) \leq \min(d_{k,q} + \Delta_k - v, e_k) - (d_{k,q} - v) \cdot u_k$ . On the other hand, if  $v > d_{k,q}$ , then the lag of  $T_k$  at  $v$  is given by the sum of the work pending for  $T_{k,q}$  (which is at most  $\min(d_{k,q} + \Delta_k - v, e_k)$ ) and the total allocation to  $T_k$  in  $\text{PS}_\tau$  in  $[d_{k,q}, v]$ . In  $\text{PS}_\tau$ ,  $T_k$  is allocated at most a fraction  $u_k$  in every instant in  $[d_{k,q}, v]$ . Hence, in this case too,  $\text{lag}(T_k, v, \mathcal{S}) \leq \min(d_{k,q} + \Delta_k - v, e_k) + (v - d_{k,q}) \cdot u_k$ . Finally, because  $v \leq d_{k,q} + \Delta_k$ , we have  $\min(d_{k,q} + \Delta_k - v, e_k) + (v - d_{k,q}) \cdot u_k \leq e_k + \Delta_k \cdot u_k$ . ■

**Lemma 8** *Let  $\delta_h \leq e_h$  denote the amount of time that the carry-in job, if any, in  $\Psi_H$  of task  $T_h$  in  $\tau_H$  executes for before  $t_d$ . Then,  $\text{LAG}(\Psi, t_d, \mathcal{S}) \leq x \cdot U_L + U_H + E_L + \sum_{T_h \in \tau_H} \delta_h \cdot (1 - u_h)$ .*

**Proof:** Let  $[t, t')$  denote the latest non-blocking, non-busy interval before  $t_d$  across which LAG increases. If  $[t, t')$  exists, then by Lemma 6,  $\text{LAG}(\Psi, t', \mathcal{S}) \leq x \cdot U_L + U_H + E_L$ . If  $[t, t')$  does not exist, then let  $t'$  be equal to the first blocking instant in  $[0, t_d)$ , if any. Otherwise, let  $t' = t_d$ . Then,  $\text{LAG}(\Psi, t', \mathcal{S}) \leq 0$ .

If no blocking, non-busy interval follows  $t'$  (by our assumption,  $[t, t')$  is the latest non-blocking interval before  $t_d$  across which LAG increases), then by Lemma 2,  $\text{LAG}(\Psi, t_d, \mathcal{S}) \leq \text{LAG}(\Psi, t', \mathcal{S})$ , completing the proof. So assume that some blocking, non-busy interval follows  $t'$ .

Let  $[t_i, t'_i)$ , where  $1 \leq i \leq b$  and  $t_i < t'_{i-1}$

for all  $1 < i \leq b$ , denote the  $b$  disjoint (*i.e.*, non-overlapping) blocking, non-busy subintervals in  $[t', t_d)$  such that the following holds: any job of  $\Psi_H$  that executes in any of the  $b$  non-busy subintervals executes continuously in the interval. It is straightforward to show that such subintervals can be defined—see Fig. 8.

Any increase in LAG for  $\Psi$  only occurs across a blocking interval after  $t'$ . By Lemma 7, the increase in LAG across the blocking subinterval  $[t_i, t'_i)$  is at most  $\sum_{T_h \in \alpha_i} (t'_i - t_i) \cdot (1 - u_h)$ , where  $\alpha_i$  is the subset of all tasks in  $\tau_H$  whose carry-in jobs are executing continuously in  $[t_i, t'_i)$ . Let  $T_h$  be a task of  $\tau_h$  whose carry-in job  $T_{h,c_i}$  executes in  $[t_i, t'_i)$ . Then, the increase in LAG due to  $T_h$  across  $[t_i, t'_i)$  is at most  $(t'_i - t_i) \cdot (1 - u_h)$ . By the statement of the lemma,  $T_{h,c_i}$  does not execute for more than  $\delta_h$  time units in  $[t', t_d)$ . Hence, the increase due to  $T_h$  over all the blocking subintervals cannot be more than  $\delta_h \cdot (1 - u_h)$ , and the increase due to all the tasks in  $\tau_H$  cannot be more than  $\sum_{T_h \in \tau_H} (1 - u_h) \cdot \delta_h$ . The lemma then follows from the upper bound for LAG at  $t'$  determined in the beginning of the proof. ■