Fair Integrated Scheduling of Soft Real-time Tardiness Classes on Multiprocessors *

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

Abstract

Prior work on Pfair scheduling has resulted in three optimal multiprocessor scheduling algorithms, and one algorithm, EPDF, that is less expensive but not optimal. EPDF is still of interest in soft real-time systems, however, due to its ability to guarantee bounded tardiness. In particular, it has been shown that a tardiness bound of t quanta is possible under EPDF if all task weights (*i.e.*, shares or utilizations) are restricted to a value specified as a function of t. In an actual system, however, different tasks may be subject to different tardiness bounds. If such a system is scheduled under EPDF, then the tardiness of a task with a higher bound may cause the tardiness bound of a task with a lower bound to be violated; that is, *temporal isolation* among the various tardiness classes may not be guaranteed. In this paper, we propose an algortihm based on EPDF for scheduling task classes with different tardiness bounds on a multiprocessor. Our algorithm provides temporal isolation among classes, allows the available processing capacity to be fully utilized, and does not require that previously established per-task weight restrictions be made more stringent.

* Work supported by NSF grants CCR 9988327, ITR 0082866, CCR 0204312, and CCR 0309825.

1 Introduction

Pfair scheduling, originally introduced by Baruah *et al.* [4], is the only known way of optimally scheduling recurrent real-time tasks on multiprocessors. Under Pfair scheduling, each task must execute at an approximately uniform rate, while respecting a fixed-size allocation quantum. A task's execution rate is defined by its *weight* (or *utilization*). Uniform rates are ensured by subdividing each task T into quantum-length *subtasks* that are subject to intermediate deadlines. To avoid deadline misses, ties among subtasks with the same deadline must be broken carefully. In fact, tie-breaking rules are crucial when devising optimal Pfair scheduling algorithms.

As discussed by Srinivasan and Anderson [9], overheads associated with tie-breaking rules may be unnecessary or unacceptable for many soft real-time systems. A soft real-time task differs from a hard real-time task in that its deadlines may sometimes be missed. If a job (*i.e.*, task instance) or a subtask with a deadline at time d completes executing at time t, then it is said to have a *tardiness* of $\max(0, t-d)$. As discussed in [6], the results produced by a soft real-time job are of decreasing usefulness after its deadline. Thus, an implicit bound exists on the tardiness that such a job can tolerate.

Systems with quality-of-service requirements, such as multimedia applications, are examples where bounded deadline misses may be tolerable. Here, fair resource allocation is necessary to provide service guarantees, but occasional deadline misses often result in tolerable performance degradation. Hence, an extreme notion of fairness that precludes all deadline misses is usually not required.

In dynamic systems that permit tasks to join or leave, the overhead introduced by tie-breaking rules may be unacceptable. In such a system, spare processing capacity may become available. To make use of this capacity, task weights must be changed on-the-fly. It is possible to reweight each task so that its next subtask deadline is preserved [9]. If no tie-breaking information is maintained, such an approach entails very little overhead. However, weight changes can cause tie-breaking information to change, so if tie-breaking rules are used, reweighting may necessitate a $\Omega(N \log N)$ cost for N tasks, due to the need to re-sort the scheduler's priority queue. This cost may be prohibitive if load changes are frequent.

The observations above motivated Srinivasan and Anderson to consider the viability of scheduling soft real-time applications using the simpler *earliest-pseudo-deadline-first* (EPDF) Pfair algorithm, which uses no tie-breaking rules. They succeeded in showing that EPDF can guarantee a tardiness of k quanta for every subtask of a feasible task system, in which each task's weight is at most $\frac{k}{k+1}$ [9]. In recent work [5], we showed that this condition can be improved to $\frac{k+1}{k+2}$. With either condition, the greater the tardiness allowed, the less stringent the weight restriction.

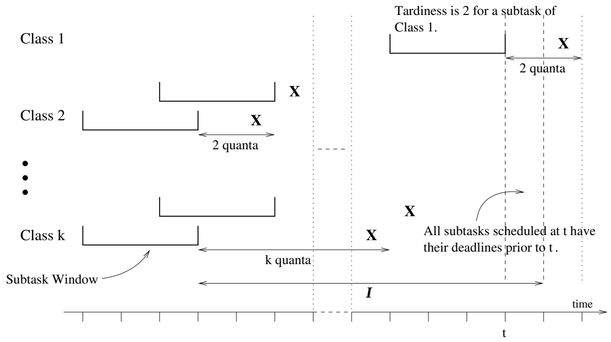


Figure 1. Under Pfair scheduling, each of a task's subtasks has an associated *window* in which it *should* be scheduled; the end of a subtask's window is its deadline. In this figure, a schedule for different classes of soft-real-time tasks on M processors under EPDF is depicted. Tasks in Class i are allowed to miss their deadlines by up to i quanta. For clarity, only a few subask windows have been shown; for each subtask shown, an 'X' denotes where it is scheduled. At time t, more than M subtasks of Classes 2 and higher with deadlines prior to t have not yet been scheduled. As a result, a subtask of Class 1 with a deadline at t cannot be scheduled at t, and hence, misses its deadline by two quanta, *i.e.* its miss threshold is exceeded.

Contributions. In the work summarized above, *all* tasks are assumed to have equal tolerance to tardiness. However, as discussed in [6], the usefulness of results produced by different soft real-time applications may decrease with tardiness at different rates; thus, different applications can be expected to have different tardiness bounds. To support multiple bounds, different tardiness classes must be *temporally isolated* from one another so that deadline misses in one class do not cause tardiness bounds to be exceeded in other classes. Preserving temporal isolation is especially important when multiplexing separately developed applications onto a multiprocessor. (Temporal isolation is a key virtue of fair scheduling.)

The tardiness bound that can be guaranteed to a task system under EPDF depends on the largest task weight. Hence, if tasks with varying tardiness bounds and weights are present in a system and are scheduled using EPDF, then it may not be possible to guarantee every task its bound. As illustrated in Fig. 1, breaking deadline ties in favor of tasks with more stringent tardiness bounds also may not be helpful. An obvious next solution would be to partition the tasks into classes by their tardiness bounds and schedule each class independently on disjoint sets of processors. Unfortunately, if the total utilization of a class is not integral, then this approach will lead to wasted processing capacity. For example, consider a task system comprised of two tardiness classes with utilizations $M_1+\delta$ and $M_2+1-\delta$, respectively, where $0 < \delta < 1$, $M_1 + M_2 + 1 = M$, and M is integral. Under partitioning, $M_1 + M_2 + 2 = M + 1$ processors will be required to schedule the two classes. Thus, processing capacity equivalent to a full processor would be wasted. In general, with q tardiness classes, $q - 1$ additional processors may be required.

In this paper, we propose a new algorithm, based on EPDF, for supporting classes with different tardiness requirements. Our algorithm provides temporal isolation among classes, allows all available processing capacity to be fully utilized, and does not require that previously established per-task weight restrictions be made more stringent. Our algorithm is described in Sec. 3 after first giving needed definitions in Sec. 2. An experimental evaluation of it is presented in Sec. 4.

2 Pfair Scheduling

In this section, we summarize the concepts of Pfair scheduling and some prior results from [4, 2, 3, 1, 8]. To begin with, we limit attention to periodic tasks, each of which begins execution at time 0. A periodic task T with an integer *period* $T.p$ and an integer *execution cost* $T.e$ has a *weight* $wt(T) = T.e/T.p$, where $0 < wt(T) < 1$. A task is *light* if its weight is less than $1/2$, and *heavy* otherwise.

Pfair algorithms allocate processor time in discrete quanta; the time interval $[t, t+1)$, where $t \in \mathcal{N}$ (the set of nonnegative integers) is called *slot* t. (Hence, time t refers to the beginning of slot t.) A task may be allocated time on different processors, but not in the same slot (*i.e.*, interprocessor migration is allowed but parallelism is not). The sequence of allocation decisions over time defines a *schedule* S. Formally, $S : \tau \times \mathcal{N} \mapsto \{0, 1\}$, where τ is a task set. $S(T, t) = 1$ iff T is scheduled in slot t. On M processors, $\sum_{T \in \tau} S(T, t) \leq M$ holds for all t.

Lags and subtasks. The notion of a Pfair schedule is defined by comparing such a schedule to an ideal fluid schedule, which allocates $wt(T)$ processor time to task T in each slot. Deviation from the fluid schedule is formally captured by the concept of *lag*. Formally, the *lag of task T at time t* is $lag(T, t) = wt(T) \cdot t - \sum_{u=0}^{t-1} S(T, u)$. A schedule is defined to be *Pfair* iff

$$(\forall T, t :: -1 < lag(T, t) < 1). \tag{1}$$

Informally, the allocation error associated with each task must always be less than one quantum. (For conciseness, we leave the schedule implicit and use $lag(T, t)$ instead of $lag(T, t, S)$.)

The lag bounds above have the effect of breaking each task T into an infinite sequence of quantum-length *subtasks*, $T_1, T_2, \ldots$. Each subtask has a *pseudo-release* $r(T_i)$ and a *pseudo-deadline* $d(T_i)$, where

$$r(T_i) = \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \ \wedge \ d(T_i) = \left\lceil \frac{i}{wt(T)} \right\rceil. \tag{2}$$

(For brevity, we often omit the prefix "pseudo-.") To satisfy (1), T_i must be scheduled in the interval $w(T_i) = [r(T_i), d(T_i))$, termed its *window*. For example, in Fig. 2(a), $r(T_1) = 0$ and $d(T_1) = 2$. Therefore, T_1 must be scheduled at either time 0 or time 1.

Soft real-time scheduling. The notion of tardiness discussed in Sec. 1 for soft real-time jobs can be extended in a straightforward manner to subtasks of soft real-time tasks. The *tardiness of a subtask* T_i is defined as $tardiness(T_i) = \max(0, t - d(T_i))$, where t is the time that T_i completes execution. The *tardiness of a task system* is then defined as the maximum tardiness among all of its subtasks in any schedule [9].

The earliest-pseudo-deadline-first (EPDF) algorithm [9] is the algorithm that we consider for scheduling soft tasks, for the reasons discussed in the introduction. EPDF prioritizes subtasks by their deadlines, and resolves any ties arbitrarily. Although EPDF is not optimal on more than two processors [1], as discussed earlier, it can ensure a tardiness of at most $k \geq 1$ quanta for each subtask, provided certain per-task weight restrictions hold.

Task models. In this paper, we consider the *intra-sporadic* (IS) and the *generalized-intra-sporadic* (GIS) task models [3, 8], which provide a general notion of recurrent execution that subsumes that found in the well-studied periodic and sporadic task models. The *sporadic* model generalizes the periodic model by allowing jobs to be released "late"; the IS model generalizes the sporadic model by allowing subtasks to be released late, as illustrated in Fig. 2(b). That is, an IS task is obtained by allowing a task's windows to be shifted right from where they would appear if the task were periodic. Let $\theta(T_i)$ denote the offset of subtask T_i, *i.e.*, the amount by which $w(T_i)$ has been shifted right. Then, by (2), we have the following.

$$r(T_i) = \theta(T_i) + \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \ \wedge \ d(T_i) = \theta(T_i) + \left\lceil \frac{i}{wt(T)} \right\rceil \tag{3}$$

The offsets are constrained so that the separation between any pair of subtask releases is at least the separation between those releases if the task were periodic. Formally,

$$k > i \Rightarrow \theta(T_k) \geq \theta(T_i). \tag{4}$$

Each subtask T_i has an additional parameter $e(T_i)$ that specifies the first time slot in which it is eligible to be scheduled. It is assumed that $e(T_i) \leq r(T_i)$ and $e(T_i) \leq e(T_{i+1})$ for all $i \geq 1$. Additionally, no subtask can become eligible before its predecessor completes execution, *i.e.*,

$$h < i \ \wedge \ e(T_i) < r(T_i) \ \wedge \ S(T_h, u) = 1 \Rightarrow \ u < e(T_i). \tag{5}$$

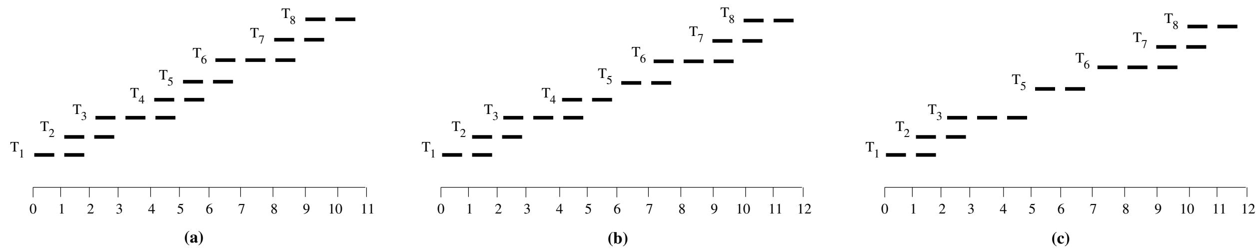


Figure 2. **(a)** Windows of the first job of a periodic task T with weight $8/11$. This job consists of subtasks $T_1, \ldots, T_8$, each of which must be scheduled within its window, or else a lag-bound violation will result. (This pattern repeats for every job.) **(b)** The Pfair windows of an IS task. Subtask T_5 becomes eligible one time unit late. **(c)** The Pfair windows of a GIS task. Subtask T_4 is absent and T_6 is one time unit late.

The interval $[r(T_i), d(T_i))$ is called the *PF-window* of T_i and the interval $[e(T_i), d(T_i))$ is called the *IS-window* of T_i. A schedule for an IS system is *valid* iff each subtask is scheduled in its IS-window. (Note that the notion of a job is not mentioned here. For systems in which subtasks are grouped into jobs that are released in sequence, the definition of e would preclude a subtask from becoming eligible before the beginning of its job.)

The IS model is suitable for many applications in which processing steps may be jittered. For example, in an application that processes packets arriving over a network, packets may arrive late or in bursts. The IS model treats these possibilities as first-class concepts: a late packet arrival corresponds to an IS delay, and if a packet arrives early (as part of a bursty sequence), then its eligibility time will be less than its Pfair release time. Note that its Pfair release time determines its deadline. Thus, in effect, an early packet arrival is handled by postponing its deadline to where it would have been had the packet arrived on time.

Generalized intra-sporadic task systems. A *generalized* intra-sporadic (GIS) task system is like an IS task system, except that a task may omit some of its subtasks. Specifically, a task T, after releasing subtask T_i, may release subtask T_k, where $k > i+1$, instead of T_{i+1}, with the following restriction: $r(T_k) - r(T_i)$ is at least $\left\lfloor \frac{k-1}{wt(T)} \right\rfloor - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor$. In other words, $r(T_k)$ is not smaller than what it would have been if $T_{i+1}, T_{i+2}, \ldots, T_{k-1}$ were present and released as early as possible. For the special case where T_k is the first subtask released by T, $r(T_k)$ must be at least $\left\lfloor \frac{k-1}{wt(T)} \right\rfloor$. Fig. 2(c) shows an example. If T_i is the most recently released subtask of T, then T may release T_k, where $k > i$, as its next subtask at time t, if $r(T_i) + \left\lfloor \frac{k-1}{wt(T)} \right\rfloor - \left\lfloor \frac{i-1}{wt(T)} \right\rfloor \le t$. If a task T, after executing subtask T_i, releases subtask T_k, then T_k is called the *successor* of T_i and T_i is called the *predecessor* of T_k.

As shown in [3], an IS or GIS task system τ is feasible on M processors iff

$$\sum_{T \in \tau} wt(T) \le M. \tag{6}$$

Shares and lags in IS and GIS task systems. $lag(T,t)$ is defined for IS and GIS tasks as before [8]. Let $ideal(T,t)$ denote the processor share that T receives in an ideal fluid (processor-sharing) schedule in $[0,t)$. Then,

$$lag(T,t) = ideal(T,t) - \sum_{u=0}^{t-1} S(T,u). \tag{7}$$

Towards defining $ideal(T,t)$, we define $share(T,u)$, which is the share assigned to task T in slot u. $share(T,u)$ is defined in terms of a function $f(T_i,t)$ that indicates the share assigned to subtask T_i in slot t. $f(T_i,t)$ is defined as follows.

$$\begin{cases} \left(\left\lfloor \frac{i-1}{wt(T)} \right\rfloor + 1\right) \times wt(T) - (i-1), & \text{if } t = r(T_i) \\ i - \left(\left\lceil \frac{i}{wt(T)} \right\rceil - 1\right) \times wt(T), & \text{if } t = d(T_i) - 1 \\ wt(T), & \text{if } t \in (r(T_i), d(T_i) - 1) \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

Fig. 3 shows some f values for a task of weight $5/16$. Given f, $share(T,u)$ can be defined as

$$share(T,u) = \sum_i f(T_i,u). \tag{9}$$

As shown in Fig. 3, $share(T,u)$ usually equals $wt(T)$, but in certain slots, it may be less than $wt(T)$. Thus,

$$(\forall T, t \ge 0 : share(T,t) \le wt(T)). \tag{10}$$

We can now define $ideal(T,t)$ as $\sum_{u=0}^{t-1} share(T,u)$. Hence, from (7),

$$lag(T,t+1) = \sum_{u=0}^{t} (share(T,u) - S(T,u))$$

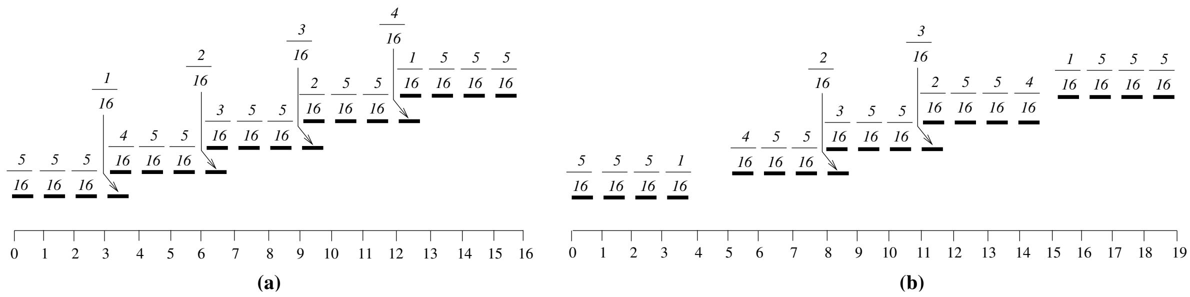


Figure 3. Fluid schedule for the first five subtasks ($T_1, \ldots, T_5$) of a task T of weight $5/16$. The share of each subtask in each slot of its PF-window is shown. In **(a)**, no subtask is released late; in **(b)**, T_2 and T_5 are released late. Note that $share(T, 3)$ is either $5/16$ or $1/16$ depending on when subtask T_2 is released.

$$= \quad lag(T,t) + share(T,t) - S(T,t). \tag{11}$$

Similarly, the total lag for a schedule S and task system τ at time $t+1$, denoted $LAG(\tau, t+1)$, is as follows. ($LAG(\tau, 0)$ is defined to be 0.)

$$LAG(\tau, t{+}1) = LAG(\tau, t) + \sum_{T \in \tau} (share(T,t) - S(T,t)). \tag{12}$$

3 Integrating Tardiness Classes

In this section, we present Algorithm I-EPDF, which schedules a soft real-time task system τ comprised of tasks with different tardiness bounds. We let tasks that can be guaranteed a tardiness of c quanta comprise Class c. Thus, if every task can be guaranteed a tardiness of at most q (≥ 1), then there are at most q classes. Any class, except Class q may be empty. If M denotes the total utilization of τ, then I-EPDF scheduled τ on at most $\lceil M \rceil$ processors. Without loss of generality, we assume that M is integral. If necessary, this property can be ensured by adding a dummy task of weight $\lceil M \rceil - M$ to τ.

The algorithm consists of three phases: **(i)** a classification phase, **(ii)** a distribution phase, and **(iii)** a scheduling phase. In the classification phase, the tardiness class of each task is identified, based on its weight. As already mentioned, Srinivasan and Anderson established a per-task weight restriction of $\frac{k}{k+1}$ for ensuring a tardiness of k quanta under EPDF [9], which we later improved to $\frac{k+1}{k+2}$ [5]. In this paper, we assume that task weights are restricted for each class using the $\frac{k}{k+1}$ condition, as the proof is simpler. Our goal in this paper is only to illustrate the idea of integrated scheduling. Our approach is still correct when the $\frac{k+1}{k+2}$ condition is used.

Classification phase. We include in Class c all tasks with weights in the range $(\frac{c-1}{c}, \frac{c}{c+1}]$, *i.e.*, tasks that can be ensured a tardiness of c quanta under EPDF. Note that this has the advantage that a task T with $wt(T) \leq \frac{c}{c+1}$ that can tolerate a tardiness of $d > c$ can be assigned to Class c and guaranteed a lower tardiness bound, without impacting the tardiness of other tasks. Letting τ^c denote the set of all tasks in Class c, this classification ensures the following property.
(W) $(\forall T \in \tau^c : \frac{c-1}{c} < wt(T) \leq \frac{c}{c+1})$
Property (W) can be ensured in $\Theta(N)$ time by simply placing task T in Class $\left\lceil \frac{T.e}{T.p - T.e} \right\rceil$.

We denote the total utilization of τ^c by M^c. Thus,

$$\tau = \bigcup_{c=1}^{q} \tau^c, \;\; M^c = \sum_{T \in \tau^c} wt(T), \text{ and } M = \sum_{c=1}^{q} M^c. \tag{13}$$

Distribution phase. The goal of this phase is to distribute the M processors among the classes and to define how processors are shared. The processors are divided into q groups of sizes $P_1, \ldots, P_q$, with the i^{th} group assigned to Class i. Because the number of processors assigned to Class i is integral, whereas its total utilization M^i may not be, each class is allowed to borrow processing capacity from at most one lower-indexed class. To ensure correctness for each class, this borrowing is subject to a number of rules given below. Later, in Sec. 3.2, we present an algorithm for defining an assignment that satisfies these rules. Before stating these rules, we introduce some relevant notation.

Notation. w^i denotes the amount of processing capacity that Class i borrows from some lower-indexed class. Sup_i denotes the lower-indexed class that supplies to Class i; if $w^i = 0$, then $Sup_i = 0$. f^i denotes the fractional part of the utilization of τ^i, *i.e.*,

$$f^i = M^i - \lfloor M^i \rfloor. \tag{14}$$

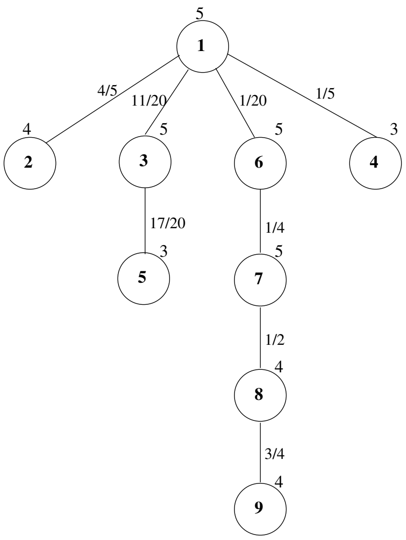
To enable the different classes to share processors at runtime, a donor task D^j ($1 \leq j \leq q$) of weight $w^j > 0$ may be created; D^j is added to Class i, where $i = Sup_j$. (The manner in which D^j is used to share processors is explained in Sec. 3.1.) The set of all donor tasks added to Class i is

Class (i)	τ^i	M^i	w^i	Sup_i	λ^i	$\hat{M}^i$	P_i
1	τ^1	$3\frac{2}{5}$	0	0	$\{D^2, D^3, D^4, D^6\}$	5	5
2	τ^2	$4\frac{4}{5}$	$\frac{4}{5}$	1	$\emptyset$	$4\frac{4}{5}$	4
3	τ^3	$4\frac{7}{10}$	$\frac{11}{20}$	1	$\{D^5\}$	$5\frac{11}{20}$	5
4	τ^4	$3\frac{1}{5}$	$\frac{1}{5}$	1	$\emptyset$	$3\frac{1}{5}$	3
5	τ^5	$3\frac{17}{20}$	$\frac{17}{20}$	3	$\emptyset$	$3\frac{17}{20}$	3
6	τ^6	$4\frac{8}{10}$	$\frac{1}{20}$	1	$\{D^7\}$	$5\frac{1}{20}$	5
7	τ^7	$4\frac{3}{4}$	$\frac{1}{4}$	6	$\{D^8\}$	$5\frac{1}{4}$	5
8	τ^8	$3\frac{3}{4}$	$\frac{1}{2}$	7	$\{D^9\}$	$4\frac{1}{2}$	4
9	τ^9	$4\frac{3}{4}$	$\frac{3}{4}$	8	$\emptyset$	$4\frac{3}{4}$	4

(a)

D^i	w^i
D^1	0
D^2	$\frac{4}{5}$
D^3	$\frac{11}{20}$
D^4	$\frac{1}{5}$
D^5	$\frac{17}{20}$
D^6	$\frac{1}{20}$
D^7	$\frac{1}{4}$
D^8	$\frac{1}{2}$
D^9	$\frac{3}{4}$

(b)



(c)

Figure 4. **(a)** Distribution of processors to the nine tardiness classes of a soft real-time task system. Column headings refer to various terms mentioned in the text. **(b)** Weights of donor tasks. **(c)** Tree representation of the task system in (a). Labels within nodes indicate class indices. The integer adjacent to a node denotes the number of processors assigned to the class that the node represents. Edge (a, b) defines the supplier/borrower relation beween Classes a and b; if $a < b$, then Class a supplies a processing capacity of $w(a, b)$ to Class b, where $w(a, b)$ is the weight of (a, b).

denoted λ^i. $\hat{\tau}^i$ extends τ^i by including these tasks:

$$\hat{\tau}^i = \tau^i \cup \lambda^i. \tag{15}$$

Correspondingly, we define

$$\hat{M}^i = M^i + \sum_{T \in \lambda^i} wt(T). \tag{16}$$

Processor sharing rules.[1] The sharing of processors among classes is governed by the following rules.

(R1) The processing capacity that Class i borrows is at most the fractional part of its utilization, *i.e.*,

$$0 \le w^i \le f^i. \tag{17}$$

(R2) Class i borrows processing capacity from at most one class with a lower tardiness bound (or a lower-indexed class), and lends to zero or more classes. In other words, the following hold.

$$(\forall i : i \ge 1 :: Sup_i < i) \tag{18}$$
$$(\forall i : i \ge 1 :: \{j \mid D^i \in \lambda^j\} = \{Sup_i\}) \tag{19}$$
$$(\forall i :: |\{j : Sup_j = i\}| \ge 0) \tag{20}$$

[1]**Some of these rules are somewhat technical in nature. They are included to address certain cases that arise in showing that I-EPDF is correct.**

(R3) Class i, where $i \ge 3$ and $f^i \le 2/3$, does not lend any processing capacity to other classes. If $0 < f^i \le 1/2$, then Class i borrows a capacity of f^i from Class 1; if f^i ranges between 1/2 and 2/3, then it borrows f^i from Class 2. If $f^i = 0$, then Class i does not borrow.

$$(\forall i : i \ge 3 \ \wedge \ f^i = 0 :: w^i = 0 \ \wedge \ Sup_i = 0) \tag{21}$$
$$(\forall i : i \ge 3 \ \wedge \ 0 < f^i \le 1/2 :: w^i = f^i \ \wedge \ Sup_i = 1 \ \wedge \ (\forall j : Sup_j \ne i)) \tag{22}$$
$$(\forall i : i \ge 3 \ \wedge \ 1/2 < f^i \le 2/3 :: w^i = f^i \ \wedge \ Sup_i = 2 \wedge \ (\forall j : Sup_j \ne i)) \tag{23}$$

(R4) The processing capacity that Class 3 or higher borrows is less than what it lends, *i.e.*,

$$(\forall i : i \ge 3 :: (\forall j : Sup_j = i \Rightarrow w^i < w^j)). \tag{24}$$

(R5) The number of processors assigned to the various classes must satisfy the following.

$$(\forall i : P_i = M^i - w^i + \sum_{\{j : Sup_j = i\}} w^j) \tag{25}$$
$$P_1 = \lceil M^1 + w^2 + \textstyle\sum_{\{j : ((j \ge 3) \ \wedge \ (f^j \le 1/2))\}} w^j \rceil \tag{26}$$
$$P_2 = \lfloor M^2 + \textstyle\sum_{\{j : ((j \ge 3) \ \wedge \ (1/2 < f^j \le 2/3))\}} w^j \rfloor \tag{27}$$
$$(\forall i : i \ge 3 :: P_i = \lceil M^i \rceil \vee \ \lfloor M^i \rfloor) \tag{28}$$
$$\sum_{i=1}^{q} P_i = M \tag{29}$$

```
ALGORITHM I-EPDF($\tau$)
    $\mathcal{E}_1, \ldots, \mathcal{E}_q$: integer; /* to denote the number of
            tasks eligible at time $t$ in each class */
    $P_1, \ldots, P_q$: integer;
    $D^1, \ldots, D^q$: Tasks;
    $\tau^1, \ldots, \tau^q$: GIS task sets;
    $\hat{\tau}^1, \ldots, \hat{\tau}^q$: GIS task sets;
    $\lambda^1, \ldots, \lambda^q$: GIS task sets initially $\emptyset$; /* Set of all
            donor tasks added to a class */
    $Sup_1, \ldots, Sup_q$: integer initially 0;
    $tight_1, \ldots, tight_q$: boolean initially TRUE

   Classification Phase
1  Group tasks into at most $q$ tardiness classes. Task sets
   $\tau^i$, where $1 \le i \le q$, are known at the end of this phase.

   Distribution Phase
2  Determine $w^i$ and create $D^i$. Determine $\lambda^i$,
   $\hat{\tau}^i$, and $P_i$, for all $i \ge 1$.

   Scheduling Phase
3  $t := 0$;
4  while TRUE do
5    for $i := q$ downto 1 do
6       if $\hat{\tau}^i \ne \emptyset$ then
7         for each $D^c$ in $\lambda^i$ do
8            if $\mathcal{E}_c \le P_c$ then
9              $s :=$ index of next eligible subtask of $D^c$;
10             if $r(D^c_s) \le t \wedge d(D^c_s) > t + 1$ then
11                if $r(D^c_s) < t$ then $s := 1$ fi;
12                $r(D^c_s),\ e(D^c_s) := t + 1,\ t + 1$
               fi
            fi
         od;
13       $\mathcal{E}_i :=$ # of eligible tasks in $\hat{\tau}^i$, excluding those
              that are early-released (it suffices to
              determine if $P_i + 1$ are eligible)
       fi
     od;
14   for $i := 1$ to $q$ do
       /* The lowest-indexed class is always tight */
15     if $tight_i$ then
16       $maxSchedulable := P_i$
       else
17       $maxSchedulable := P_i + 1$
       fi;
18     Schedule at most $maxSchedulable$ tasks of
       $\hat{\tau}^i$ using EPDF (for Class 3 and higher, break ties
       involving the heavy donor task, if any,
       in favor of the donor task);
19     for each $D^c$ in $\lambda^i$ do
20       if $D^c$ is scheduled then
21            $tight_c :=$ FALSE
         else
22            $tight_c :=$ TRUE
         fi
       od;
23     $t := t + 1$
     od
```

Figure 5. Algorithm I-EPDF—detailed pseudo-code for the scheduling phase.

The supplier/borrower relationship among classes can be represented as a weighted tree in which nodes represent classes. An edge of weight w between nodes i and j, where $i < j$, implies that $w^j = w$ and $Sup_j = i$. As an example, Fig. 4 shows an assignment of processors to classes and a supplier/borrower relation among classes that conforms to the rules above for a task system comprised of nine tardiness classes.

The following properties follow from (R1)–(R5).

(L1) There are at most two tasks in $\hat{\tau}^1$ with weights exceeding $1/2$.
(L2) The weight of every task in $\hat{\tau}^2$ is in $(\frac{1}{2}, \frac{2}{3}]$.
(L3) $(\forall i : i \ge 3 \wedge \lambda^i \ne \emptyset \Rightarrow f^i > \frac{2}{3}$.)

3.1 Scheduling Phase of I-EPDF

Assuming that processors are assigned to classes per the rules above, we now explain the scheduling phase. A proof that it is correct is given in Appendix C. As mentioned earlier, Sec. 3.2 presents an algorithm for creating such an assignment. In the scheduling phase, a separate instantiation of EPDF is used to schedule each $\hat{\tau}^i$. The pseudo-code for this phase is shown in lines 3–23 in Fig. 5. Note that $\hat{\tau}^i$ includes the donor tasks in λ^i, which compete with tasks in τ^i at every time instant. If at time t, a donor task D^j in λ^i is scheduled, then one of the processors of Class i is handed down to Class j. Thus, Class j has $P_j + 1$ processors for scheduling the tasks in $\hat{\tau}^j$ at time t, zero of more of which may be handed down to higher-indexed classes, recursively.

In the first part of the scheduling phase, given by the **for** loop of lines 5–13, the number of eligible subtasks of $\hat{\tau}^c$ at time t is identified, when $i = c$. Because the **for** loop considers the classes in decreasing index order, the number of eligible tasks in classes with higher indices than c are known at this time. Therefore, if $\hat{\tau}^c$ includes donor task D^k and the number of eligible tasks in $\hat{\tau}^k$ is at most P_k, then the release time of the next subtask D^k_s of D^k is postponed to $t + 1$, if its deadline is greater than $t + 1$. We do this because Class k is not able to use an extra processor that it would be given, and hence, by postponing the release time of the next subtask of its donor task under the conditions specified, Class k may be provided with an extra processor sooner in the future than may otherwise be possible. We refer to this scheduling rule in lines 8–12 as the *postponement* rule. Another related rule in line 11 is that if the release time of D^k_s before the postponement was earlier than t, then D^k_s is replaced by D^k_1 with $r(D^k_1)$ set to $t + 1$. This rule shall be referred to as the *reset* rule. As discussed later, this rule does not impact the tardiness of other tasks.

The second part of the scheduling phase, given by the

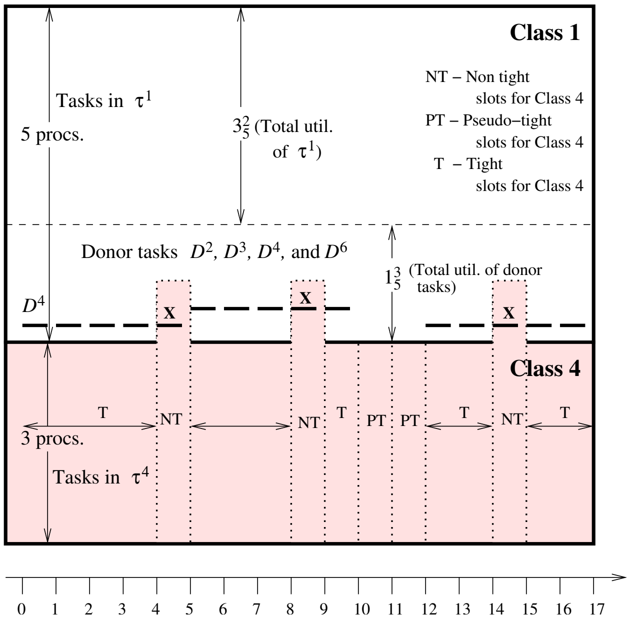


Figure 6. Classes 1 and 4 of the example task system in Fig. 4. Class 1 is assigned five processors and supplies processing capacity to Classes 2, 3, 4, and 6. Class 4 is assigned three processors and borrows a processing capacity of $1/5$ from Class 1. An additional processor is handed down to Class 4 from Class 1 when donor task D^4 is scheduled in Class 1. The example partial schedule shows the first three subtasks of D^4, which are scheduled in the slots marked by an 'X'. The release of the third subtask is postponed from time 10 to time 12. Thus, slots 4, 8, and 14 are non-tight for Class 4, slots 10 and 11 are pseudo-tight (see the appendix), and the rest are tight.

for loop in lines 14–23, determines the maximum number of tasks of $\hat{\tau}^c$ that can be scheduled ($maxSchedulable$) at t, and schedules those with the highest priority. For all but the lowest-indexed class, $maxSchedulable$ is either P_c or $P_c + 1$, based on whether D^c is scheduled. If P_c processors are available for scheduling tasks in $\hat{\tau}^c$, then t is said to be a *tight* slot for $\hat{\tau}^c$; otherwise, t is a *non-tight* slot for $\hat{\tau}^c$. An example is given in Fig. 6, in which Classes 1 and 4 from Fig. 4 are considered.

Selecting the highest priority tasks to schedule dominates the per-slot time complexity of this phase, and hence, it is the same as that of EPDF, *i.e.*, $O(M \log N)$.

3.2 Distribution Phase of I-EPDF

(R1)–(R5) can be expressed as linear constraints and the problem solved using integer or mixed integer programming. (Floors and ceilings in the expressions can easily be eliminated.) However, as we now show, a solution in linear time is possible. Fig. 7 presents the detailed pseudo-code for the distribution phase. In describing this code, we refer to Class j as an *i-borrower* if Class j borrows processing capacity from Class i. The computation here proceeds in three steps. The **for** loop in lines 0–0 comprises the first step, which is responsible for ensuring that (R3) holds, and together with the third step (see below), that (28) is satisfied. In this step, Class i, where $i \geq 3$, is set to borrow its entire fractional utilization of f^i from Class 1, if $f^i \leq 1/2$, or from Class 2, if $1/2 < f^i \leq 2/3$. This is done by setting $w^i = M^i - \lfloor M^i \rfloor = f^i$ in line 0, followed by the addition of a donor task D^i of appropriate weight to Class 1 or Class 2. Every class that is made a 1- or a 2-borrower at the end of Step 1, is considered *finished*, and does not participate in future distribution steps. This is marked by setting the boolean variable $done_i$ to TRUE. Such a class is assigned $\lfloor M^i \rfloor$ processors, which are not shared with other classes. For example, consider Fig. 4. The total utilization of τ^4 is $3\frac{1}{5}$, with a fractional part $f^4 = 1/5 < 1/2$. Therefore, at the end of the step just described, a donor task D^4 of weight $w^4 = 1/5$ is added to Class 1, three processors are assigned to Class 4, and it is marked finished.

Lines 8–12 comprise the next step, which ensures (27). In this step, Class 2 is made a 1-borrower by letting it borrow a processing capacity of $w^2 = (M^2 + \sum_{T \in \lambda^2} wt(T)) - \lfloor (M^2 + \sum_{T \in \lambda^2} wt(T)) \rfloor$ from Class 1. It is then marked finished. In the example in Fig. 4, the fractional part of the utilization of no class is between $1/2$ and $2/3$. Therefore, no donor tasks are added to Class 2 in the previous step. Thus, $w^2 = 4/5$, a donor task D^2 of weight $4/5$ is added to Class 1, and Class 2 is marked finished.

The **while** loop in lines 14–40 constitutes the third step in the distribution phase. This step is responsible for ensuring (24), (26) and (28). In this step, every class that is not yet finished is considered in increasing index order. The goal of the i^{th} iteration is to determine at most two higher-indexed classes with which Class i can share its spare capacity (given by $spare_i = \lceil \hat{M}^i - w^i \rceil - (\hat{M}^i - w^i)$). (To ensure the tardiness bound of the borrowing class, it is necessary to ensure that it does not borrow from a class with a larger bound.) Class i is also marked finished at the end of the i^{th} iteration. Thus, at the beginning of iteration i, every class with a lower index than i is already finished. Note that for Class 3 and higher, $\hat{M}^i = M^i$ holds at the beginning of the iteration in which it is considered. This is because these classes are not augmented with donor tasks prior to this point. Line 19 identifies Class l with the lowest index greater than i that is not finished that can be made an i-borrower. To ensure (R1), if $f^l \leq spare_i$ holds, then Class l is made to borrow a processing capacity of f^l from Class i and is marked finished in the **if** block in lines 21–27. Line 27 identifies and sets l to the next higher-indexed class that is not yet finished.

Irrespective of whether the test in line 21 succeeded, at line 28, $f^l > avail$ holds. This is clearly the case if the test in line 21 failed. On the other hand, if this test succeeded,

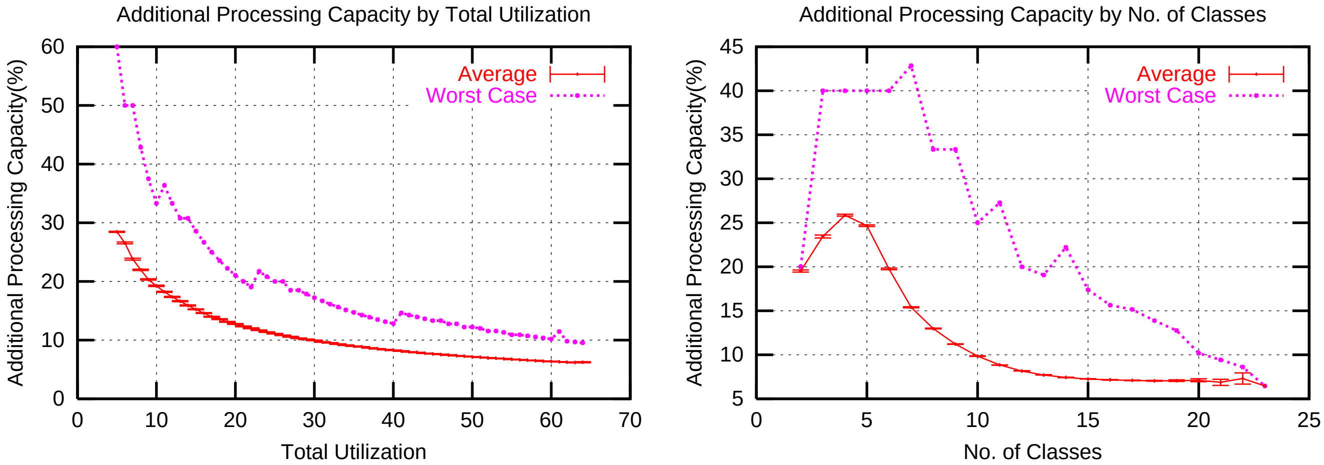
```
PROCEDURE AddDonorTask(i, w, sup)
1   D^i := donor task of weight w;
2   Sup_i := sup;
3   τ̂^sup, λ^sup := τ̂^sup ∪ {D^i}, λ^sup ∪ {D^i};
4   M̂^sup := M̂^sup + w^i;
5   return

ALGORITHM I-EPDF(τ)
        P_1, ..., P_q: integer;
        w^1, ..., w^q: rational;
        D^1, ..., D^q: GIS tasks;
        τ^1, ..., τ^q: GIS task sets;
        τ̂^1, ..., τ̂^q: GIS task sets;
        λ^1, ..., λ^q: GIS task sets initially ∅; /* Set of all
                       donor tasks added to a class */
        Sup_1, ..., Sup_q: integer initially 0;
        spare_1, ..., spare_q: rational initially 0;
        done_1, ..., done_{q+1}: boolean initially FALSE

    Classification Phase
    /* Group tasks into at most q tardiness classes. Task sets
    τ^i, where 1 ≤ i ≤ q are known at the end of this phase. */

    Distribution Phase
1   for i := 3 to q do
2     if M^i − ⌊M^i⌋ ≤ 2/3 then
3         w^i := M^i − ⌊M^i⌋;
4         if w^i > 0 then
5           if w^i ≤ 1/2 then Sup_i := 1 else Sup_i := 2 fi
6           AddDonorTask(i, w^i, Sup_i)
          fi;
7         P_i, done_i := ⌊M^i⌋, TRUE
      fi
    od;
8   w^2 := M̂^2 − ⌊M̂^2⌋;
9   if w^2 > 0 then AddDonorTask(2, w^2, 1) fi;
10  P_2, done_2 := ⌊M̂^2⌋, TRUE;
11  if ⌈M̂^1⌉ = M̂^1 then
12    P_1, done_1 := M̂^1, TRUE
    fi;
13  i := 1;
14  while i ≤ q do
      /* done_{q+1} handles the boundary condition */
15    while done_i do i := i + 1 od;
16    if i ≤ q then
17        avail := spare_i := ⌈M̂^i − w^i⌉ − (M̂^i − w^i);
18        l := i + 1;
19        while done_l do l := l + 1 od;
20        if l ≤ q then
21          if avail > 0 ∧ (M^l − ⌊M^l⌋) ≤ avail then
22              w^l := M^l − ⌊M^l⌋;
23              AddDonorTask(l, w^l, i);
24              P_l, done_l := ⌊M^l⌋, TRUE;
25              avail := avail − w^l;
26              l := l + 1;
27              while done_l do l := l + 1 od
          fi;
28        if avail > 0 /* ∧ l ≤ q */ then
29              w^l := avail;
30              AddDonorTask(l, w^l, i);
31              d, j := l, i;
32              while w^d < w^j do
33                τ̂^j, λ^j := τ̂^j − {D^d}, λ^j − {D^d};
34                w^j, M̂^j := w^j − w^d, M̂^j − w^d;
35                τ̂^{Sup_j} := τ̂^{Sup_j} ∪ {D^d};
36                λ^{Sup_j} := λ^{Sup_j} ∪ {D^d};
37                if w^j < w^d then d := j fi;
38                j := Sup_j
              od
          fi;
39        P_i, done_i := ⌊M̂^i⌋, TRUE;
        fi
40      i := l
    fi
    od
```

Figure 7. Algorithm I-EPDF—detailed pseudo-code for the distribution phase.

then because $f^l > 2/3$ holds for every class of index three or higher that is not finished by Step 2, $w^l > 2/3$ holds at line 25. Because $spare_i$, as computed in line 17, is less than one, $avail < 1/3$ holds at the end of line 25. Hence, for the same reason that $f^l > 2/3$ holds for every class of index exceeding three that is not finished by Step 2, $f^l > avail$ holds at line 28. Because the amount of processing capacity that Class l borrows is set to $\min(avail, f^l)$, Class i can have at most two donor tasks added to it in the i^{th} iteration. l is the unfinished class with the lowest index at line 40. Therefore, i is updated to l so that Class l is considered for the addition of donor tasks in the next iteration.

Using our example (Fig. 4), $spare_1$ at the beginning of the first iteration of the **while** loop in lines 14–40 is $\lceil \hat{M}^1 \rceil - \hat{M}^1 = 5 - 4\frac{2}{5} = \frac{3}{5}$. (Because D^4 of weight $w^4 = \frac{1}{5}$ and D^2 of weight $w^2 = \frac{4}{5}$ were added to Class 1 in Steps 1 and 2, respectively, $\hat{M}^1 = M^1 + w^4 + w^2 = 3\frac{2}{5} + \frac{1}{5} + \frac{4}{5} = 4\frac{2}{5}$, and hence, $\lceil \hat{M}^1 \rceil = 5$, at the end of Step 2.) In this iteration, the first unfinished class with a higher index than one, which is Class 3, is made a 1-borrower (lines 28–30). Thus, $l = 3$ in this case. Hence, at the end of the first iteration, w^3 is set to $3/5$ (line 29) and a donor task D^3 of weight w^3 is added to Class 1 (line 30). Class 1 is then marked finished (line 39). The unfulfilled utilization of Class 3 is now $4\frac{7}{10} - \frac{3}{5} = 4\frac{1}{10}$. Therefore, Class 3 has a spare capacity $spare_3 = \frac{9}{10}$. Because Class 3 is the next unfinished class, classes with which its spare capacity is shared are identified in the next iteration.

One final adjustment is performed in lines 32–38. If the weight of the donor task w^l that is added to Class i is less than w^i, *i.e.*, the processing capacity that Class i borrows in turn from its supplier $j = Sup_i$, then Class j is made Class l's supplier, too. This is done by moving D^l to Class j from Class i. w^i is appropriately reduced so that the total capacity that Class j supplies remains the same. As a result of this adjustment, Class j will now have two donor tasks



(a)

(b)

Figure 8. Empirical determination of additional processing capacity, expressed as a percentage of total utilization, that may be required if I-EPDF is not used to schedule soft real-time task sets comprised of multiple classes. **(a)** Additional capacity vs. Total utilization. **(b)** Additional capacity vs. Number of classes.

D^i and D^l in place of D^i (it is possible that Class j has some other donor tasks, whose weights are not altered in this iteration), and it is possible for one of them to be lighter than D^j. If this is the case, then the **while** loop in lines 32–38 moves the lighter of the two donor tasks, D^i and D^l, up the supplier chain, to ensure (R4).

In our example, as described earlier, $spare_3 = \frac{9}{10}$ at the end of the iteration for Class 1. This holds at the beginning of the iteration for Class 3, when $i = 3$. Recall that Class 4 is already finished, and hence, Class 5 is the next unfinished class. Because $f^5 = \frac{17}{20}$, which is less than $spare_3$, the code in lines 21–27 makes Class 5 borrow the entire fractional part of its utilization from Class 3, and reduces $avail$ to $\frac{9}{10} - \frac{17}{20} = \frac{1}{20}$ (in line 25). The next unfinished class is identified to be 6 in line 27. Hence, $l = 6$ at line 28, and the code in lines 28–30 makes Class 6 borrow a capacity of $\frac{1}{20}$ from Class 3. Thus, at the end of line 30, donor tasks D^5 and D^6, of weights $w^5 = \frac{17}{20}$ and $w^6 = \frac{1}{20}$, respectively, are added to Class 3. Recall that Class 3 already borrows a processing capacity of $\frac{3}{5}$ from Class 1. Therefore, $w^3 = \frac{3}{5}$, and hence, $w^6 < w^3$. As a result, D^6 is moved to Class 1, the supplier of Class 3, and the weight of D^3 (w^3) is reduced by w^6 to $\frac{11}{20}$. In other words, at the end of the iteration for Class 3, Class 3 borrows only $\frac{11}{20}$ from Class 1 and is augmented with only one donor task, D^5. Fig. 4 shows the final distribution and sharing of processors among classes.

It can be shown that the complexity of the above algorithm is $\Theta(q)$. A proof that it ensures (R1)–(R5) is presented in Appendix B.

4 Experimental Evaluation

In this section, we report results of our empirical evaluation of the additional processing capacity that may be required, when classes do not share processors. The evaluation procedure was as follows. 1,000,000 task sets were generated at random, with total utilization M in the range $5..64$. The tasks in each task set were divided into q tardiness classes based on their weights. The total number of processors P required to schedule the task set was then computed, assuming that each tardiness class has exclusive access to the processors that it requires. The difference $E = P - M$, which represents the additional processing capacity required, was then determined. The average value of E (expressed as a percentage of M) with respect to total utilization (M) and the number of classes (q) is plotted in Fig. 8. 99% confidence intervals are also shown on these plots. The figure also depicts the worst-case observed values of E, for each value of M and q. The graphs show that the average percentage of loss is quite high (over 30%), for small values of M and q, and decreases with increasing M and q. The reason for this is as follows. In Fig. 8(a), the value of q for a given M is the average over all task sets with that value of M, and in Fig. 8(b), the value of M for a given q is the average over all task sets with that value of q. As mentioned in the introduction, E is at most $q - 1$. Also, because a class can span multiple processors, q increases at a lower rate than M. Therefore, E, expressed as a percentage of M, decreases with increasing q and M. (However, for small q, q may increase at a higher rate than M because the minimum value of M is 5, while that of q is 2. This explains the initial rise in Fig. 8(b).) Even though E decreases as M and q increase, the loss is still more than 5% for large M and q, which suggests significant waste.

5 Conclusion

We have presented a new algorithm for integrating soft real-time tardiness classes on a multiprocessor. Our algorithm provides temporal isolation among classes, allows available processing capacity to be fully utilized, and does not require that previously established per-task weight restrictions for a given tardiness threshold be lowered. Our experiments indicate that the proposed algorithm allows a substantial amount of processing capacity to be reclaimed.

The algorithm presented could be extended to allow hard tasks, in addition to soft tasks. In that case, an optimal algorithm (with tie breaks) is used for scheduling hard tasks, while a separate instantiation of EPDF is used for each soft class. However, it may be required to promote a few soft tasks to the hard category.

As discussed in Sec. 1, one motivation for using EPDF is the ability to reweight tasks efficiently in dynamic systems. However, reweighting a task may alter the tardiness bound that can be guaranteed to it, and hence, may require that the task be migrated to a different tardiness class. Redistributing processors to the redefined classes can be done in constant time. It only remains to be proved that the tardiness bounds of individual tasks can still be guaranteed. We are currently working on this problem.

Acknowledgements: We are grateful to Phil Holman for his suggestions on improving the presentation of this paper and for his comments on earlier drafts.

References

[1] J. Anderson and A. Srinivasan. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. *Journal of Computer and System Sciences*. To appear.

[2] J. Anderson and A. Srinivasan. Early-release fair scheduling. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, pages 35–43, June 2000.

[3] J. Anderson and A. Srinivasan. Pfair scheduling: Beyond periodic task systems. In *Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications*, pages 297–306, Dec. 2000.

[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] U. Devi and J. Anderson. Improved conditions for bounded tardiness under EPDF fair multiprocessor scheduling. In Submission, November 2003.

[6] J.W.S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[7] A. Srinivasan. *Efficient and Flexible Fair Scheduling of Real-time Tasks on Multiprocessors*. PhD thesis, University of North Carolina at Chapel Hill, December 2003.

[8] A. Srinivasan and J. Anderson. Optimal rate-based scheduling on multiprocessors. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 189–198, May 2002.

[9] A. Srinivasan and J. Anderson. Efficient scheduling of soft real-time applications on multiprocessors. In *Proceedings of the 15th Euromicro Conference on Real-time Systems*, pages 51–59, July 2003.

A Pfair Scheduling – Additional Properties

In this appendix we state additional properties of pfair scheduling that is needed for the correctness proof of I-EPDF presented in Appendix C.

The first three lemmas concern lengths of windows of subtasks.

Lemma 1 [**1**] *The length of each window of a task T is either $\lceil \frac{1}{wt(T)} \rceil$ or $\lceil \frac{1}{wt(T)} \rceil + 1$.*

Lemma 2 [**1**] *The following properties hold for any task T.*

(**a**) *If $(i-1)$ is a multiple of $T.e$, then $|w(T_i)| = \left\lceil \frac{1}{wt(T)} \right\rceil$.*

(**b**) *If $b(T_i) = 0$, then $|w(T_i)| = |w(T_{i+1})|$.*

(**c**) *If $b(T_i) = 0$, then $w(T_i)$ is a minimal window of T.*

(**d**) *If T is heavy and $b(T_i) = 0$, then $|w(T_i)| = 2$.*

Lemma 3 *If the length of the window of a subtask T_i of T is k, then $wt(T) \geq 1/k$.*

Proof: By Lemma 1, $|w(T_i)| = \left\lceil \frac{1}{wt(T)} \right\rceil$ or $|w(T_i)| = \left\lceil \frac{1}{wt(T)} \right\rceil + 1$. Therefore, $\left\lceil \frac{1}{wt(T)} \right\rceil = k$ or $\left\lceil \frac{1}{wt(T)} \right\rceil + 1 = k$. If the former holds, then $wt(T) \geq \frac{1}{k}$, while if the latter holds, then $wt(T) \geq \frac{1}{k-1}$, which implies that $wt(T) \geq \frac{1}{k}$. $\square$

The next lemma summarizes some general properties of the f[2] values of (8).

Lemma 4 [**7**] *Let f be as defined by* (8)*. Then, the following hold.*

(**a**) *In any time slot $u \geq 0$, at most two consecutive subtasks of a task may have positive values for f.*

(**b**) *If $f(T_i, r(T_i)) < wt(T)$, then $b(T_{i-1}) = 1$.*

(**c**) *If $f(T_i, d(T_i) - 1) < wt(T)$, then $b(T_i) = 1$.*

(**d**) *If $b(T_{i-1}) = 1$ and T_i exists, then $f((T_{i-1}, d(T_{i-1}))) + f(T_i, r(T_i)) = wt(T)$.*

[2] The f value of a subtask is also referred to as the flow that the subtask receives in an ideal fluid schedule.

B Correctness Proof for the Distribution Phase

In this appendix, we state and prove lemmas concerning properties that hold at the end of the distribution phase of Algorithm I-EPDF. For conciseness, by I-EPDF, we refer to its distribution phase in this section.

All references to line numbers are with respect to the pseudo-code in Fig. 7.

Lemma 5 *Donor tasks are not added to Class c, where $c \geq 3$, before the iteration of the* **while** *loop in lines 14–40, in which $i = c$.*

Proof: Inspection of code in lines 1–12 shows that donor tasks are not added to Class c in that part. The **while** loop in lines 14–40 considers classes in increasing index order. (The value of i for the next iteration of the loop is updated to l in line 40, and it can be verified that $l > i$.) In the i^{th} iteration, new donor tasks are added only to Class i in lines 23 and 30. The **while** loop in lines 32–38 moves donor tasks across classes, but only across those classes that are already augmented with such tasks. Therefore, donor tasks are not added to a class with an index greater than i. □

Lemma 6 *If donor task D^c is created, then $done_c =$ FALSE holds prior to its creation.*

Proof: Donor tasks are created in lines 6, 9, 23, and 30. We consider each case.

The **for** loop in lines 1–7 considers Class 3 and higher in increasing index order exactly once. Because the *done* array is initialized to FALSE, and is not altered for Class c until after D^c is created (which is done at most once) $done_c =$ FALSE holds before the creation of D^c.

In line 9, D^2 is created. It can easily be verified that $done_2 =$ FALSE at this time.

In lines 23 and 30, donor tasks D^l are created, where l is determined in lines 19 and 27 such that $done_l =$ FALSE holds. □

Lemma 7 *If $i = c$, where $c \leq q$, holds for an iteration of the* **while** *loop in lines 14–40, then $done_c = true$ holds at the end of the iteration.*

Proof: Within the **while** loop, the value of i is modified in lines 15 and 40 only. If modified in line 15, then $done_c =$ TRUE already holds. Before i is updated in line 40 for the next iteration, $done_c$ is updated to TRUE in the previous line (line 39). Note that 39 is always executed if $c \leq q$. □

Lemma 8 *D^c, where $c \geq 1$, is created at most once.*

Proof: By Lemma 6, donor task D^c is created only if $done_c =$ FALSE holds prior to its creation. Therefore, it suffices to show that $done_c$ is updated to TRUE either immediately after the creation of D^c, or before an attempt can be made to create it again.

Donor tasks are created in lines 6, 9, 23, and 30. If created in line 6, 9, or 23, $done_c$ is updated to TRUE in the subsequent line. If created in line 30, then $l = c$ holds in that line. It can be verified that i is updated to l, *i.e.*, c in line 40 for the next iteration of the **while** loop.By Lemma 7, $done_c =$ TRUE will hold at the end of the next iteration. It can also be verified that in an iteration of the **while** loop referred to, D^i is not created. Therefore, D^c will not be created for a second time in the future. □

Lemma 9 *D^c is created and added to a lower-indexed class before donor tasks are added to Class c, where $c \geq 3$.*

Proof: Follows from Lemmas 5, 6, and 7, and the fact that D^c is not created in that iteration of the **while** loop in lines 14–40, for which $i = c$ holds. □

Lemma 10 *If D^c is created in line 23 or 30, then $c \geq 3$ and $f^c > 2/3$.*

Proof: D^1 is never created (or does not exist), D^2 is created in line 9, and if $c \geq 3 \wedge f^c \leq 2/3$, then D^c is created in line 6. By Lemma 8, a donor task is created exactly once. The lemma follows from these facts. □

Lemma 11 *Let $w^c = f^c$, where $c \geq 3$. Then, Class c is not augmented with donor tasks.*

Proof: D^c, where $c \geq 3$, may be created in lines 6, 23, or 30. We consider two cases.

Case 1: D^c is created in line 6 or 23. In this case, $w^c = f^c$, and $done_c =$ TRUE holds immediately after. By Lemma 9, Class c is not augmented with donor tasks before D^c is created, and by Lemma 5, the augmentation occurs only in the iteration of the **while** loop in lines 14–40, for which $i = c$ holds. However, because $done_c =$ TRUE holds, by line 15, Class c is skipped from consideration in the **while** loop, and hence, is not augmented with donor tasks.

Case 2: D^c is created in line 30. For this case, we first show that $w^c < f^c$ holds at the time of creation of D^c. Consider the iteration of the **while** loop in which D^c is created in line 30. Prior to the creation of D^c, a different donor task D^d may have been created in line 23in the same iteration. We consider both possibilities. If no other donor task was created, then the test in line 21should have failed. Because D^c is created in line 30, the test in line 28should have succeeded, which implies that $M^l - \lfloor M^l \rfloor = f^l > avail$ holds. Because w^l is set to $avail$ in line 29before D^l is created, by $c = l$, it implies that $f^c > w^c$ holds. On the other hand, if

D^d was created in line 23prior to the creation of D^c in line 30we reason as follows. The value of $avail$ as computed in line 17is at most one. By Lemma 10, $f^d \geq 2/3$, and because w^d is set to f^d in line 22and $avail$ is reduced by w^d in line 25, $avail < 1/3$ holds at line 29. Again, by Lemma 10, $f^c \geq 2/3$, whereas w^c is set to $avail$, which is less than $1/3$. It can be verified that the only piece of code that may alter the weights of the donor tasks is the **while** loop in lines 32–38, and that the weights are only decreased. Therefore, w^c can never equal f^c. □

Lemma 12 $0 \leq w^c \leq f^c$, *where* $c \geq 1$.

Proof: w^c is initialized to zero, modified when D^c is created, and may later be modified in the **while** loop in 32–38. By Lemma 8, D^c is created exactly once. Inspection of the code shows that D^c is created in lines 6, 9, 23, or 30. If created in lines 6, 9, or 23, $w^c = f^c$. Therefore, it remains to be shown that $w^c \leq fc$ holds, if D^c is created in line 30.

By Lemma 10, if D^c is created in line 30, then

$$c \geq 3 \ \wedge \ f^c > 2/3. \tag{30}$$

We consider execution from the beginning of the iteration of the **while** loop in lines 32–38 in which D^c is created. It is easy to see that $avail = spare_i \leq 1$ holds at the end of line 17. If the test in line 21 succeeded, then another donor task D^d, where $d < c$ is created in line 23 before D^c in the same iteration. By Lemma 10, $f^d > 2/3$. Because $w^d = f^d$ holds by the assignment in line 22and $avail$ is updated to $avail - w^d$ is line 25, $avail < 1/3$ holds at line 28. Therefore, by (30), $f^c > avail$ holds at line 28. If the test in line 21 failed, while that in line 28 succeeded, then it is easy to see the $f^c > avail$ holds. w^c is set to $avail$, which by the argument just concluded is less than f^c, before D^c is created in line 30.

Once set to a non-zero value, the value of w^c is modified only in the **while** loop in lines 32–38. However, the value is never increased, and if decreased, the reduction is by an amount that is strictly less than its present value. Therefore, if set to a non-zero value, $0 < w^c \leq f^c$ holds. If never set, then $w^c = 0$ holds. □

Lemma 13 *Let Class c, where* $c \geq 3$*, be augmented with donor tasks. Then, the value of* $spare_c$*, as computed in line* 17*, is equal to* $1 - f^c + w^c$*, which is greater than* w^c.

Proof: $spare_c$ as computed in line 17 is given by $\lceil \hat{M}^c - w^c \rceil - (\hat{M}^c - w^c)$. If $c \geq 3$, then by Lemma 5, Class c is not augmented with donor tasks until the iteration of the **while** loop in lines 14–40. Therefore, by (16), at line 17, when $i = c$, $\hat{M}^c = M^c$. Therefore, $spare_c$ is given by

$$\begin{aligned}
&\lceil M^c - w^c \rceil - (M^c - w^c) \\
&= \lceil \lfloor M^c \rfloor + f^c - w^c \rceil - (\lfloor M^c \rfloor + f^c - w^c) \text{ (from (14))} \\
&= \lfloor M^c \rfloor + \lceil f^c - w^c \rceil - \lfloor M^c \rfloor - f^c + w^c \\
&= \lceil f^c - w^c \rceil - f^c + w^c \\
&= 1 - f^c + w^c.
\end{aligned}$$

The final step is by Lemmas 11 and 12. If $spare_c$ is less than or equal to w^c, then it would imply that $w^c \geq 1 - f^c + w^c$, or that $f^c \geq 1$, which is a contradiction to (14). □

Lemma 14 *The sum of the weights of the donor tasks added to Class c in that iteration of the* **while** *loop in lines* 14– 40 *for which* $i = c$ *is exactly equal to* $spare_c$*, as computed in line* 17 *either at the end of line* 27 *or line* 30.

Proof: The proof follows from the fact that M is integral. □

Lemma 15 $w^1 = 0$.

Proof: Straightforward. □

Lemma 16 *If the sum of the weights of the donor tasks added to Class* j *equals* $1 - f^j + w^j$ *prior to the execution of an iteration of the* **while** *loop in lines* 32–38*, then the same relation holds at the end of the iteration.*

Proof: At the beginning of the iteration, we have $\sum_{\{k:Sup_k=j\}} w^k = 1 - f^j + w^j$. Within the **while** loop, one donor task D^d is moved from Class j to Class Sup_j. Therefore, $\sum_{\{k:Sup_k=j\}} w^k = 1 - f^j + w^j - w^d$, at the end of the iteration. However, w^j is reduced by w^d to $w^j - w^d$ in the same iteration. Therefore, $\sum_{\{k:Sup_k=j\}} w^k = 1 - f^j + w^j$, still holds at the end of the iteration. □

Lemma 17 *The values of* $\lfloor \hat{M}^j \rfloor$ *and* $\hat{M}^{Sup^j}$ *at the end the execution of an iteration of the* **while** *loop in lines* 32–38 *are the same as their values at the beginning of that iteration.*

Proof: By Lemmas 15 and 12, and the test in line 32, we have $j \neq 1$. It can also be easily shown that $j \neq 2$, and hence, $j \geq 3$ holds. Therefore, by Lemmas 14, 13, and 16, we have

$$\sum_{\{k:Sup^k=j\}} w^k = 1 - f^j + w^j. \tag{31}$$

We first show that $\lfloor \hat{M}^j \rfloor$ is not altered. $\lfloor \hat{M}^j \rfloor$ at the beginning of an iteration of the loop is given by

$$\begin{aligned}
\lfloor \hat{M}^j \rfloor &= \lfloor M^j + \textstyle\sum_{\{k:Sup^k=j} w^k \} \rfloor \\
&= \lfloor M^j + 1 - f^j + w^j \rfloor \quad \text{(from (31))} && (32) \\
&= \lfloor \lfloor M^j \rfloor + 1 + w^j \rfloor \quad \text{(from (14))} && (33) \\
&= \lfloor M^j \rfloor + 1. && (34)
\end{aligned}$$

Within the **while** loop, D^d is moved from Class j to Class Sup_j. Therefore, by (31), the sum of the weights of the donor tasks present in Class j at the end of the iteration is given by $\sum_{\{k:Sup^k=j\}} w^k = 1 - f^j + w^j - w^d$. Hence, $\lfloor \hat{M}^j \rfloor$ at the end of the iteration is given by

$$\begin{aligned}
\lfloor \hat{M}^j \rfloor &= \lfloor M^j + \textstyle\sum_{\{k:Sup^k=j} w^k\} \rfloor \\
&= \lfloor M^j + 1 - f^j + w^j - w^d \rfloor && (35) \\
&= \lfloor \lfloor M^j \rfloor + 1 + w^j - w^d \rfloor \quad \text{(from (14))} && (36) \\
&= \lfloor M^j \rfloor + 1 + \lfloor w^j - w^d \rfloor && (37) \\
&= \lfloor M^j \rfloor + 1. \ \text{(because } w^j > w^d\text{)} && (38)
\end{aligned}$$

(34) and (38) imply that the value of $\lfloor \hat{M}^j \rfloor$ is not altered by the execution of the **while** loop.

To see that the value of $\hat{M}^{Sup_j}$ is not altered, note that D^d with weight w^d is moved to Sup_j, while the weight of an existing donor task, D^j, is reduced by w^d. Therefore, the sum of the weights of the donor tasks present in Sup_j is not altered at the end of the iteration, and therefore, by (16), $\hat{M}^{Sup_j}$ is not altered. □

Lemma 18 $P_1 = \hat{M}^1$.

Proof: The value of P_1 is set in line 12 or 39. If assigned in line 12, P_1 takes a value of $\hat{M}^1$, and the lemma holds. Therefore, in the rest of the proof we assume that P_1 is set in line 39. By Lemma 15, $w^1 = 0$ holds. Therefore, $spare_1$, as computed in line 17, in the iteration of the **while** loop in lines 14–40 for which $i = 1$ equals $\lfloor \hat{M}^1 \rfloor - \hat{M}^1$. By Lemma 14, the sum of the weights of the donor tasks added to Class 1 in the **while** loop's iteration when $i = 1$ is exactly equal to $spare_1 = \lfloor \hat{M}^1 \rfloor - \hat{M}^1$. Therefore, at line 39, $\hat{M}^1$ equals $\hat{M}^1$ as of line 17, plus $spare_1$, *i.e.*, $\lfloor \hat{M}^1 \rfloor$ as of line 17. Thus, $\hat{M}^1$ is integral at line 39, and hence, $\lfloor \hat{M}^1 \rfloor = \hat{M}^1$, and the assignment in line 39 ensures that $P_1 = \hat{M}^1$.

Inspection of the code in lines 14–40 shows that $i = 1$ holds only for one iteration of the **while** loop in those lines. Therefore, P_1 is assigned exactly once. After the assignment, $P_1 = \hat{M}^1$ can fail to hold only if donor tasks are added or removed after the assignment. Again, no newly created donor tasks can be added to Class 1 after the first iteration of the **while** loop. However, existing donor tasks may be moved from some other class to Class 1 in the **while** loop in lines 32–38. By Lemmas 15 and 12, and the test in line 32, we have $j \neq 1$ for the **while** loop, and by Lemma 17, $\hat{M}^{Sup_j}$ is not altered by the execution in the **while** loop. Inspection of the code in the **while** loop shows that only the composition of Classes j and Sup_j may be altered. Therefore, the value of $\hat{M}^1$ is not altered after P_1 is set. □

Lemma 19 $P_c = \lfloor \hat{M}^c \rfloor$, *for* $c \geq 2$.

Proof: Similar to the proof of Lemma 18. □

Lemma 20 *The sum of the weights of the donor tasks added to Class c, where* $c \geq 3$ *is given by* $1 - f^c + w^c$.

Proof: Follows from Lemmas 5, 13, 14, and 16. □

Lemma 21 *The sum of the weights of the donor tasks added to Class c, where* $c \geq 3$*, is greater than* w^c *and is at most one.*

Proof: By Lemma 20, the sum of the weights of the donor tasks added to Class c is given by $1 - f^c + w^c$, where by Lemmas 12 and 11, $1 - f^c + w^c < 1$. By Lemma 13, this value is greater than w^c. □

Lemma 22 *If Class c, where* $c \geq 3$ *is augmented with a donor task* D^d *with weight* $w^d \geq 5/6$*, then* $w^c > 1/2$.

By (L3), $f^c > 2/3$, and by Lemma 20, the sum of the weights of the donor tasks added to Class c equals $1 - f^c + w^c$. Because $w^d \geq 5/6$ and $f^c > 2/3$, it implies that $1 - f^c + w^c \geq 5/6$, or that $w^c > 1/2$. □

Lemma 23 *If Class c, where* $c \geq 3$*, is augmented with a donor task* D^d*, then* $w^d > w^c$.

Proof Sketch: We have the following from Lemmas 5, 13, and 14, and inspection of code in the **while** loop beginning at line 14. Class c is augmented with donor tasks for the first time in that iteration of the **while** loop of line 14, in which $i = c$; the augmentation is in line 23 or 30 or both. The sum of the weights of the donor tasks added is exactly equal to $spare_c$, as computed at the beginning of the iteration in line 17, and lines 23 and 30 are executed at most once. If only one donor task is added, then the lemma follows directly from Lemma 21. Hence, the lemma may be falsified only if two donor tasks are added, one each in lines 23 and 30. We show that if a donor task D^d with $w^d < w^c$ is added to Class c in an iteration of the **while** loop of line 14, then the inner nested **while** loop in lines 32–38ensures that $w^d > w^c$ holds before the end of the iteration of the outer **while** loop, without causing any other class to have donor tasks with weights less than the weight of the donor task for that class. (In the rest of this proof, references to the **while** loop are to the inner loop.)

We consider the situation just after two donor tasks, D^u and D^v, are added (at lines 23 and 30) to Class c and before the execution of the **while** loop commences. We assume that Class c is the only task with a donor task whose weight is less than w^c. Therefore, the following holds.

$$(\forall i : i \geq 3 \land \ i \neq c :: (\forall D^j \in \lambda^i :: w^j > w^i)) \qquad (39)$$

To show that $w^d > w^c$ holds at the end of the **while** loop, we show that the following is an invariant of the loop. In the rest of this proof, the values of j and d are as dictated by the code in lines 28–38.

$$(\forall k : k \geq 3 \wedge\ k \neq j :: (\forall D^m \in \lambda^k :: w^m > w^k)) \quad (40)$$

Initialization: Before the first iteration of the **while** loop, $j = c$. Therefore, the invariant follows from (39).
Maintenance: We consider the following cases.

Case 1: $j = c$**.** $j = c$ for the first iteration of the **while** loop. Because D^u and D^v are the only donor tasks in Class c, by Lemma 21,

$$w^u + w^v > w^c. \quad (41)$$

Because D^u is added in line 23, $w^u = f^u$ holds, where $f^u > 2/3$. (Otherwise, D^u is added to Class 1 or Class 2 in lines 1–7.) Because $avail = spare_c \leq 1$ (line 17), and w^v is assigned the value of $avail$ updated to $avail - w^u$, which is less than $1/3$ in line 25, $w^v < w^u$ holds at line 30. Thus, we have $w^v < w^u$.

d is assigned a value that is equal to the index of the second donor task added to Class c in line 31. Therefore, $d = v$ holds, and within the **while** loop, D^v, which is lighter of the two tasks D^u and D^v, is removed from Class c and is added to Class Sup_c, while w^c is decreased by w^v. Thus, Class c now contains only one donor task, D^u. By (41), $w^u > w^c - w^v$, where $w^c - w^v$ is the new value of w^c. Therefore, at the end of the first iteration, Class c contains a single donor task D^u with $w^u > w^c$. However, because D^v is moved to Class Sup_c, $w^v > w^{Sup_c}$ may not hold. But then j is updated to Sup_j in line 38, and because $j = c$ held before the update, the loop invariant can be seen to hold at the end of Iteration 1.

Case 2: $j \neq c$**.** Class j, where $j \neq c$ may have a donor task with weight less than w^j only if during the previous iteration of the loop, a donor task, say D^r, were moved to to Class j, and the weight of an existing donor task, say D^s, (whose identity was j in that iteration) were reduced by w^r. Because no other donor task, other than D^s, had its weight altered, and D^r is the only new donor task moved to Class j, because (39) held prior to the execution of the **while** loop, D^r and D^s could be the only donor tasks in Class j with their weights less than w^j. The lighter of these two donor tasks, say D^r, is determined in line 37 during the previous iteration, which in turn is moved to Sup_j during this iteration. The weight of D^j is also appropriately reduced. It can now be shown easily that the weight of the other donor task, D^s, is greater than w^j. Therefore, at the end of this iteration, Class j would not contain any donor task whose weight is less than w^j. However, the class from which Class j borrows, Sup_j, may now have donor tasks with weights less than w^{Sup_j}. Because $j = Sup_j$ for the next iteration (by the assignment in line 38), the loop invariant is maintained.

Termination: When the loop terminates, $w^d > w^j$, which by the arguments for the two cases above implies that Class j does not contain any donor task with weight less than w^j. This fact, together with the loop invariant implies that

$$(\forall i : i \geq 3 :: (\forall D^j \in \lambda^i :: w^j > w^i))$$

□

Lemma 24 *Let D^m be a heavy donor task added to Class c, where $c \geq 3$. Then, D^m is the only donor task in λ^c.*

Proof: Let D^n be a second donor task added to Class c. Then, by Lemma 21, $w^n > w^c$. By Lemma 20, the sum of the weights of the donor tasks added to Class c equals $1 - f^c + w^c$. Therefore, $1 - f^c + w^c \geq w^m + w^n$. Because $w^m \geq 1/2$ and $w^n > w^c$, we have $1 - f^c + w^c > 1/2 + w^c$, or $f^c < 1/2$, which contradicts (L3). Therefore, D^m is the only donor task added to Class c. □

Lemma 25 $w^c = \hat{M}^c - \lfloor \hat{M}^c \rfloor$.

Proof: We consider the following cases.

Case 1: $c = 1$**.** By Lemma 15, $w^1 = 0$. By Lemma 18, $P_1 = \hat{M}^1$, which implies that $\hat{M}^1$ is integral. Therefore, $\hat{M}^1 = \lfloor \hat{M}^1 \rfloor$, *i.e.*, $\hat{M}^1 - \lfloor \hat{M}^1 \rfloor = 0$, and hence, the lemma holds for this case.

Case 2: $c = 2$**.** After the assignment in line 8, $w^c = \hat{M}^c - \lfloor \hat{M}^c \rfloor$ holds. It can be shown that donor tasks are neither added to Class 2 nor are their weights altered after this assignment. (Because the classes to which Class 2 supplies do not have donor tasks added to them, j will never equal the indices of those classes, and hence, 2, in the **while** loop in lines 32–38, where donor tasks are moved across classes or their weights are modified.)

Case 3: $c \geq 3 \wedge w^c = f^c$**.** In this case, by Lemma 11, donor tasks are not added to Class c. Therefore, $\hat{M}^c = M^c$, and hence, $\hat{M}^c - \lfloor \hat{M}^c \rfloor = M^c - \lfloor M^c \rfloor$, where $M^c - \lfloor M^c \rfloor = f^c$, by (14).

Case 4: $c \geq 3 \wedge w^c \neq f^c$**.** By Lemma 12, we have $w^c < f^c$. By Lemma 20, the sum of the weights of the donor tasks added to Class c is given by $1 - f^c + w^c$. Therefore, by (16),

$$\hat{M}^c = M^c + 1 - f^c + w^c, \quad (42)$$

which implies that $\lfloor \hat{M}^c \rfloor = \lfloor M^c + 1 - f^c + w^c \rfloor$, which equals $\lfloor \lfloor M^c \rfloor + 1 + w^c \rfloor$ by (14). Because $w^c < f^c$ and $f^c < 1$, we have $\lfloor \lfloor M^c \rfloor + 1 + w^c \rfloor = \lfloor M^c \rfloor + 1$. In other words,

$$\lfloor \hat{M}^c \rfloor = \lfloor M^c \rfloor + 1. \quad (43)$$

From (42), we have $w^c = \hat{M}^c - (M^c + 1 - f^c)$, which by (14) equals $\hat{M}^c - (\lfloor M^c \rfloor + 1)$. That w^c equals $\hat{M}^c - \lfloor \hat{M}^c \rfloor$ then follows from (43). □

Rules (R1)–(R5) can easily be seen to follow from the lemmas in this subsection.

C Correctness Proof for the Scheduling Phase

In this section, a complete correctness proof for the scheduling phase of Algorithm I-EPDF is presented. For brevity, we use "I-EPDF" to refer to its scheduling phase in this section. Proving that I-EPDF is correct consists of proving that tasks in τ^c are guaranteed a tardiness of c. We do this by showing that I-EPDF guarantees a tardiness of c for tasks in $\hat{\tau}^c$.

Based on the donor tasks they are augmented with, the tardiness classes that result at the end of the distribution phase may be classified as follows.

Type 1 Class c is of Type 1 iff $w^c = 0$.

Type 2 Class c is of Type 2 iff $w^c > 0$, and for all $D^d \in \lambda^c$, $w^d \leq \frac{c}{c+1}$. In other words, Class c is of Type 2, iff the weight of every donor task added to it is at most the maximum weight permissible for a tardiness of c.

Type 3 Includes every class that is not of Type 1 or Type 2.

Our proof obligation is to show that I-EPDF is correct for all the three types of classes. For notational simplicity, we show that I-EPDF correctly schedules the class with the lowest index possible in each type. The resulting arguments can easily be generalized to apply to classes with higher indices.

It can easily be verified that Class 1, if non-empty, is of Type 1. From (L2), we can conclude that Class 2, if non-empty, is of Type 1 or 2. Class 3 and higher can be of any of the three types. Therefore, we assume that Class 1, Class 2, and Class 3 are non-empty, are of Types 1, 2, and 3, respectively, and prove that I-EPDF guarantees a tardiness of one, two, and three, respectively, to these classes.

The total lag LAG of a task system, as discussed in Sec. 2, if positive, is a measure of the extra demand on the system in the future, over what it is equipped to handle. Hence, if LAG is high and work arrives at the prescribed rate in the future, then the tardiness bounds may be violated. Therefore, the proof for all the three classes primarily consists of showing that LAG does not increase to an amount that is necessary for a violation.

C.1 Tardiness Bounds for $\hat{\tau}^1$

The total utilization $\hat{M}^1$ of $\hat{\tau}^1$ (or a class of Type 1) is integral and the class is provided with $\hat{M}^1$ processors. Hence, the proof of Theorem 1 in [9], which states that if the sum of the $M - 1$ largest task weights is at most $\frac{M+1}{2}$, then a tardiness of at most one can be guaranteed, can be applied directly. Note that this implies a per-task weight restriction of $\frac{M+1}{2M-2}$, which converges to $\frac{1}{2}$, for large M. It can easily be shown that if the weights of at most two tasks exceed $\frac{1}{2}$, as is the case with $\hat{\tau}^1$, by (L1), then the sum of the $M - 1$ largest task weights is at most $\frac{M+1}{2}$. In the proof in [9], LAG is bounded by bounding the lags of individual tasks. $\hat{\tau}^1$ differs from the task systems considered there only in the nature of the subtasks of its donor tasks due to the reset rule. However, this rule neither violates the feasibility condition in (6) nor leads to an increased lag bound for D^m, and hence, correctness for Class 1 follows directly from the result in [9].

C.2 Tardiness Bounds for $\hat{\tau}^2$

By Lemma 25, $w^2 = \hat{M}^2 - \lfloor \hat{M}^2 \rfloor$, and by Lemma 19, $P_2 = \lfloor \hat{M}^2 \rfloor$. Thus, tasks in $\hat{\tau}^2$ are assured $\lfloor \hat{M}^2 \rfloor$ processors at all times. By our assumption, w^2 is non-zero, and because D^2 is added to Class 1, tasks in $\hat{\tau}^2$ will be provided with an additional processor assigned to Class 1, whenever D^2 is scheduled in $\hat{\tau}^1$. As mentioned earlier, a slot t is called *non-tight* or *tight* based on whether D^2 is scheduled in t or not. Thus, in a non-tight slot, $\lceil \hat{M}^2 \rceil$ processors are available for scheduling tasks in $\hat{\tau}^2$, and in a tight slot, $\lfloor \hat{M}^2 \rfloor$ processors are available. Our assumption that $w^2 > 0$ also implies that $\lceil \hat{M}^2 \rceil \neq \lfloor \hat{M}^2 \rfloor$.

At time t, I-EPDF postpones the release time of the next eligible subtask of a donor task to $t + 1$ in line 12, if the class that the donor task serves cannot utilize an extra processor at t. This results in tight slot t resembling a non-tight slot to the served class. We distinguish such slots by calling them *pseudo-tight*. Fig. 9 illustrates this. The following is a formal definition of pseudo-tight slots.

Definition 1: A tight slot t is a *pseudo-tight slot* for $\hat{\tau}^c$ if the release time of the next eligible subtask of D^c is postponed to $t + 1$, and thus made ineligible, at t.

There are a few other details that need explanation. First, the next eligible subtask of D^c, D^c_i, is postponed at t, only if its deadline is later than $t + 1$. Next, if prior to the postponement, the release time of D^c_i were earlier than t, then it is replaced by D^c_1.

Before considering pseudo-tight slots further, we describe the notion of an active task as introduced in [8], which is used in later lemmas that concern such slots. The share that a GIS task receives in the ideal system may be zero during certain time slots, if subtasks are absent or are released late. Tasks with and without subtasks at time t can be distinguished using the following definition of an *active* task.

Definition 2:[8] A task U is *active* at time t if it has a subtask U_j such that $e(U_j) \leq t < d(U_j)$.

The next three lemmas are concerned with pseudo-tight slots.

Lemma 26 *Let a donor task D^c be inactive over time slots $[t_1, t_2]$, and active at $t_2 + 1$. Then, t_2 is a pseudo-tight slot for $\hat{\tau}^c$.*

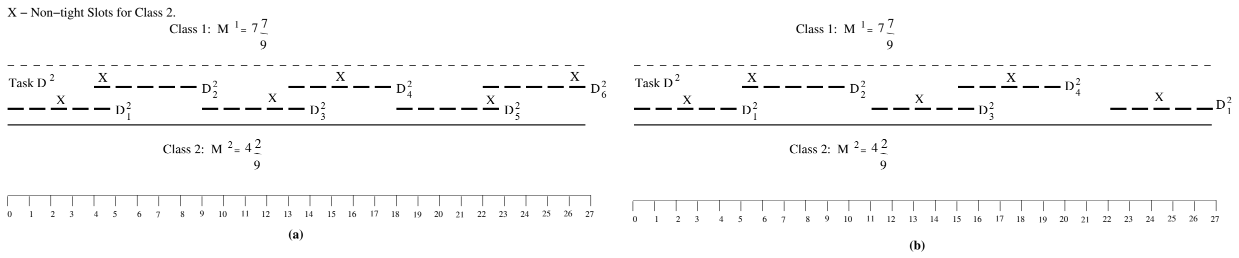


Figure 9. Partial schedule of a soft real-time system with two classes, Class 1 and Class 2, where $M^1 = 7\frac{7}{9}$ and $M^2 = 4\frac{2}{9}$, and hence, $wt(D^2) = w^2 = \frac{2}{9}$. **(a)** The windows of the first six subtasks of D^2 are depicted. No subtask of D^2 is postponed. D^2 is scheduled in time slots 2, 4, 12, 15, 22, and 26, which are marked by an 'X'. Therefore, 5 processors are available for scheduling tasks of Class 2 in these slots, which are thus "non-tight" for Class 2. The remaining slots are "tight." **(b)** The release times of subtasks D^2_2 and D^2_3, are postponed at times 4 and 10, respectively. The release time of subtask D^2_5 is postponed at time 21. Because $r(D^2_5)$ is originally 20, which is less than 21, the time of postponement, the index of the next subtask of D^2 is reset to 1. 4, 10, and 21 are therefore pseudo-tight slots for Class 2. Time slots marked by an 'X' are non-tight and the remaining slots are tight.

Proof: For D^c to be inactive at t_2 and active at $t_2 + 1$, its next eligible subtask at t_2 must have had its eligibility and release times postponed to t_2+1. Hence, t_2 is a pseudo-tight slot for $\hat{\tau}^c$. $\Box$

Lemma 27 *Let t be a pseudo-tight slot in a schedule S^c, and let D^c be active in t. Then, $0 < share(D^c, t) < wt(D^c)$ and $lag(D^c, t) = -share(D^c, t)$.*

Proof Sketch: Let D^c_j be the next eligible subtask of D^c at t, and D^c_i its predecessor. We consider the release times and deadlines of D^c_j and D^c_i before D^c_j is postponed. Because t is a pseudo-tight slot, by Def. 1, $r(D^c_j) \leq t$. Because windows of successive subtasks can overlap by at most one slot, this implies that $d(D^c_i) \leq t + 1$, and that if $d(D^c_i) = t + 1$, then $i = j - 1$. After the postponement, $e(D^c_j) = r(D^c_j) > t$. Therefore, for D^c to be active in t, there should exist some other subtask satisfying Def. 2, which by the previous discussion implies that $d(D^c_i) = t+1$ and $i = j - 1$. The alignment of the windows of D^c_{j-1} and D^c_j must be as shown in Fig. 10(e). From the figure, D^c's share in slot t equals $share(D^c_{j-1}, t)$, which by (8) is less than $wt(D^c)$.

We next show that $lag(D^c, t) = -share(D^c, t)$. Because t is a pseudo-tight slot, D^c is not scheduled at t. Because at a pseudo-tight slot t, the release time of the next eligible subtask of D^c is postponed, every preceding subtask of D^c has been scheduled. From Fig. 10(e), this implies that, at time t, D^c is ahead of the ideal system by an amount equal to D^c's share in slot t. It follows that $lag(D^c, t) = -share(D^c, t)$. $\Box$

Lemma 28 *Let $0 < share(D^c, t) < w^c$. Then $t = r(D^c_i)$ or $t = d(D^c_i) - 1$, for some $i > 0$. In the former case, D^c_i is released late, while in the latter case, D^c_{i+1} is either absent or is released late.*

Proof: Because $share(D^c, t) = \sum_i f(D^c_i, t)$, and (8) implies that $f(D^c_i, t) = w^c$ if $t \neq r(D^c_i)$ and $t \neq d(D^c_i) - 1$, we have $t = r(D^c_i)$ or $t = d(D^c_i) - 1$. Parts (b) and (c) of Lemma 4 imply that in the former case $b(D^c_{i-1}) = 1$. Therefore, if D^c_i is not released late, then $t = d(D^c_{i-1}) - 1$, which by Lemma 4(d) implies that $share(D^c, t) = w^c$, which is a contradiction. A similar contradiction can be arrived at for the latter case also, if D^c_{i+1} is present and is not released late. $\Box$

Lemma 29 *Let $share(D^c, t) < w^c$ and let $t = r(D^c_i)$, for some $i > 0$. Then D^c_i's predecessor is D^c_{i-1} and $share(D^c, d(D^c_{i-1}) - 1) = w^c - share(D^c, t)$. Also, every slot in $[d(D^c_{i-1}) - 1, t - 1]$ is pseudo-tight.*

Proof Sketch: By Lemma 28, D^c_i is definitely released late. It can be verified from lines 5–13 of Fig. 5 that I-EPDF interrupts the periodic nature of D^c in one of the following ways only. It postpones the release of the next eligible subtask, or makes D^c_1 the next subtask, releasable at the next time instant. In the latter case, it can be verified from (8) that $f(D^c_1, r(D^c_1))$ and $share(D^c, r(D^c_1))$ equal w^c. Therefore, by the statement of the lemma, it should be the case that $i \neq 1$. It can also be verified from the pseudo-code that the indices of the subtasks of D^c occur in sequence, unless reset to one. Therefore, the predecessor of D^c_i should be D^c_{i-1}. Because $share(D^c, r(D^c_i)) < w^c$, from (9) and (8) we have

$$f(D^c_i, r(D^c_i)) = share(D^c, r(D^c_i)) < w^c, \qquad (44)$$

which by Lemma 4(b) implies that

$$b(T_{i-1}) = 1. \qquad (45)$$

Therefore, by Lemma 4(d), $f(D^c_{i-1}, d(D^c_{i-1}) - 1) + f(D^c_i, r(D^c_i)) = w^c$, or by (44), $f(D^c_{i-1}, d(D^c_{i-1}) - 1) =$

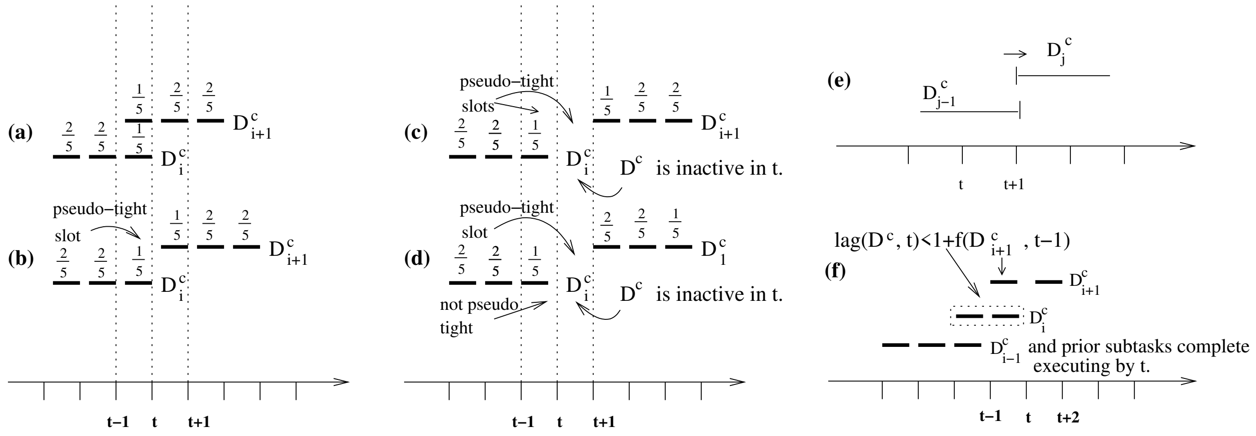


Figure 10. Two consecutive subtasks D_i^c and D_{i+1}^c of D^c with weight 2/5. Flows are shown in each slot. **(a)** There is no IS separation between D_i^c and D_{i+1}^c. The share of D^c equals its weight in all slots shown. **(b)** The release time of D_{i+1}^c is postponed at $t-1$; hence, the share of D^c is less than its weight in both $t-1$ and t, and $t-1$ is a pseudo-tight slot. **(c)** The release time of D_{i+1}^c is postponed at both $t-1$ and t; hence, D^c is inactive at t, and its share is less than its weight in $t-1$ and $t+1$. Both $t-1$ and t are pseudo-tight. **(d)** The release time of D_{i+1}^c is postponed at t, but not at $t-1$; hence, D^c is inactive at t, and its share is less than its weight only in $t-1$. In this case, only t is pseudo-tight. **(e)** Lemmas 27 and 30. **(f)** Lemma 33.

$w^c - share(D^c, r(D_i^c))$. Because no subtask other than D_i^c can overlap with the last slot of D_{i-1}^c and D_i^c is released late, the share of D^c in that slot in the ideal system is given by the flow received by D_{i-1}^c. Therefore, we have $share(D^c, d(D_{i-1}^c)) = w^c - share(D^c, r(D_i^c))$. This completes the proof for the first part of the lemma. This is also illustrated in Figs. 10(b) and (c).

To see that every slot in $[d(D_{i-1}^c) - 1, t-1]$ is pseudo-tight first observe that $t-1$ is necessarily pseudo-tight. If $d(D_{i-1}^c) = t$, then the proof is complete. Therefore, assume $d(D_{i-1}^c) < t$. Next, consider the latest slot $t' < t-1$ in the interval specified, that is not pseudo-tight. Because $b(D_{i-1}^c) = 1$ (by (45)), if D_i^c is not postponed, then $r(D_i^c) = d(D_{i-1}^c) - 1$. Also, I-EPDF postpones the release times of the subtasks of D^c by at most one slot (*i.e.*, at time t, it postpones the release time to $t+1$). Therefore, the release time of D_i^c could not have been postponed to $t'+1$ or later, prior to t'. Therefore, because t' is not pseudo-tight but $t'+1$ is, it implies that $r(D_i^c)$ as of time $t'+1$ should have been at or before t'. If that were the case, then I-EPDF would have replaced D_i^c by D_1^c (refer lines 9–12 of Fig. 5), and hence, our assumption that t' is not pseudo-tight is incorrect. Therefore, every slot in the interval specified is pseudo-tight. □

Lemma 30 *If t is a pseudo-tight slot, then $lag(D^c, t+1) = 0$.*

Proof: If t is a pseudo-tight slot, then the release time of the next releasable subtask of D^c is postponed at t, as illustrated in Fig. 10(d). Because t is pseudo-tight, where the release time of D_j^c is postponed, D_{j-1}^c must have been scheduled prior to t. Thus, by time $t+1$, both the ideal and the actual systems have scheduled all of D^c's subtasks up through D_{j-1}^c prior to $t+1$. This implies that $lag(D^c, t+1) = 0$. □

We prove that under I-EPDF the tardiness of $\hat{\tau}^2$ is at most two, by contradiction. If not, then t_d, τ, and $\hat{\tau}^2$ defined as follows exist. (In these definitions, it is assumed that for the task systems τ and σ mentioned, Classes 1 and 2 are nonempty.)

Definition 3: t_d is the earliest subtask deadline for which there is a violation under I-EPDF, *i.e.*, there exists some task system τ with a subtask in $\hat{\tau}^2$ with a deadline at t_d and a tardiness of three, and there does not exist any other task system σ with a subtask in $\hat{\sigma}^2$ with a deadline prior to t_d and a tardiness of three.

Definition 4: τ is a feasible task system with the following properties.

(T1) t_d is the earliest deadline of a subtask in $\hat{\tau}^2$ with a tardiness of three under I-EPDF.

(T2) No other feasible task system σ satisfying (T1) contains fewer subtasks of tasks in $\hat{\sigma}^2$ than τ.

(T3) No feasible task system σ satisfying (T1) and (T2) has a larger rank than τ, where the *rank* of a task system τ at t is defined as $\sum_{T \in \hat{\tau}^2} \sum_{\{T_i | d(T_i) \leq t_d\}} e(T_i)$.

By (T1) and (T2), exactly one subtask in $\hat{\tau}^2$ has a tardiness of three: if several such subtasks exist, then all but one can be removed and the remaining subtask will still have a tardiness of three, contradicting (T2). Additionally, the following assertions follow from the above properties and definitions.

(A1) $(\exists T_i \in \hat{\tau}^2 : d(T_i) = t_d \wedge \ tardiness(T_i) = 3)$
(A2) $(\forall T_i \in \hat{\tau}^2 : d(T_i) < t_d \Rightarrow tardiness(T_i) \leq 2)$

In the rest of this paper, we use S to denote an I-EPDF schedule for τ and S^c to denote that part of S that corresponds to the schedule for $\hat{\tau}^c$.

It is easy to see that the tardiness of T_i is not affected by subtasks with deadlines greater than t_d. Note that any such subtask is scheduled before t_d only if no subtask with a deadline at most t_d is eligible at that slot. Thus, by (T2), we can assume that no subtask in $\hat{\tau}^2$ has a deadline greater than t_d. In other words, the following holds.

(A3) $(\forall T_i \in \hat{\tau}^2 : d(T_i) \leq t_d)$.

We next determine the number of subtasks of $\hat{\tau}^2$ that should miss their deadlines at t_d for $\hat{\tau}^2$ to have a tardiness of three (at t_d). Unless otherwise specified, all remaining lemmas are assumed to apply to $\hat{\tau}^2$.

Lemma 31 *The number of subtasks that have their deadlines at or prior to t_d and are not scheduled by t_d is at least $2 \cdot \lfloor \hat{M}^2 \rfloor + k + 1$, where $1 \leq k \leq 2$, is the number of non-tight slots in the next two slots, t_d and $t_d + 1$.*

Proof: We prove the lemma for $k = 1$; the other cases are similar. By (A1), some subtask T_i with a deadline at t_d has a tardiness of three. It follows that this subtask is scheduled in slot $t_d + 2$. Because both t_d and $t_d + 1$ are tight, $\lfloor \hat{M}^2 \rfloor$ processors are available to the tasks in $\hat{\tau}^2$ in each of these slots. By (A3), it suffices to show that $\lfloor \hat{M}^2 \rfloor$ subtasks from $\hat{\tau}^2$ are scheduled in each of these slots. First, suppose that fewer than $\lfloor \hat{M}^2 \rfloor$ such subtasks are scheduled in slot $t_d + 1$. Then, because T_i could not be scheduled there, its predecessor must have been scheduled at $t_d + 1$. But then, T_i's predecessor would have a tardiness of at least 3, contradicting the fact that only T_i has a tardiness of three (and no subtask has greater tardiness). Having shown that $\lfloor \hat{M}^2 \rfloor$ subtasks from $\hat{\tau}^2$ are scheduled at $t_d + 1$, it follows that the same is true of t_d. Otherwise, one of the subtasks scheduled at $t_d + 1$ would have been eligible at t_d and hence, would have been scheduled there. □

From the above lemma, we have the following.

Lemma 32 *$LAG(\hat{\tau}^2, t_d)$ is at least $2 \cdot \lfloor \hat{M}^2 \rfloor + k + 1$, where $1 \leq k \leq 2$ is the number of non-tight slots in the next two slots, t_d and t_{d+1}.*

Proof: Again, we consider only when $k = 1$. The ideal system (by definition) completes the execution of every subtask in $\hat{\tau}^2$ (each of which, by (A3), has a deadline at or before t_d) by time t_d. By Lemma 31, Algorithm I-EPDF fails to schedule at least $2 \cdot \lfloor \hat{M}^2 \rfloor + 1$ of these subtasks by time t_d. Thus, $LAG(\hat{\tau}^2, t_d) \geq 2 \cdot \lfloor \hat{M}^2 \rfloor + 1$. □

The next three lemmas are concerned with the lag of D^2, the donor task of $\hat{\tau}^1$.

Lemma 33 $-1 < lag(D^2, t) < 1 + w^2$, *for all t.*

Proof: This lemma follows easily from the fact that a tardiness of one is guaranteed for Class 1. If a task T may miss its deadline by only at most one quantum, then its lag must always be strictly less than $1 + wt(T)$. The -1 lag bound in the lemma holds because each subtask of D^2 has equal eligibility and release times, *i.e.*, the IS and PF windows for such subtasks are always the same. □

Lemma 34 *If $lag(D^2, t) \geq 1$, and the next releasable subtask of D^2 is not postponed at t, then t is non-tight.*

Proof: If $lag(D^2, t) \geq 1$, then by (11), $lag(D^2, t+1) \geq 1 + share(D^2, t) - S(D^2, t)$. If there are no IS separations, then $share(D^2, t) = wt(D^2) = w^2$. Therefore, if D^2 were not scheduled at t, then $lag(D^2, t + 1) \geq 1 + w^2$, contradicting Lemma 33. Therefore, D^2 is scheduled at t, and thus t is non-tight. □

Lemma 35 *If $lag(D^2, t) \geq k - w^2$, where $1 \leq k \leq 2$, and the next releasable subtask of D^2 is not postponed at t and $t + 1$, then at least k of slots t and $t + 1$ are non-tight.*

Proof: We prove the lemma for $k = 1$. The proof for $k = 2$ is similar. If t is non-tight, the proof is complete. Therefore, assume t is tight. If $lag(D^2, t) \geq 1 - w^2$, then by (11), $lag(D^2, t+1) \geq 1 - w^2 + share(D^2, t) - S(D^2, t)$. If there are no IS separations, then $share(D^2, t) = w^2$. Therefore, $lag(D^2, t + 1) \geq 1$, which by Lemma 34 implies that $t + 1$ is non-tight. □

In what follows, we obtain the desired contradiction by showing that $LAG(\hat{\tau}^2, t_d)$ is **(i)** less than $2 \cdot \lfloor \hat{M}^2 \rfloor + 1$ if t_d and $t_d + 1$ are both tight, **(ii)** less than $2 \cdot \lfloor \hat{M}^2 \rfloor + 2$ if at most one of t_d and t_{d+1} is not non-tight, and **(iii)** less than $2 \cdot \lfloor \hat{M}^2 \rfloor + 3$ if t_d and $t_d + 1$ are both non-tight, thereby contradicting Lemma 32. We do this by establishing the following.

$$LAG(\hat{\tau}^2, t_d) \leq 2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t_d) + w^2 \qquad (46)$$

It can be verified that if (46) holds, then Lemmas 33–35 imply that $LAG(\hat{\tau}^2, t_d)$ is always less than $2 \cdot \lfloor \hat{M}^2 \rfloor + 3$, and is less than $2 \cdot \lfloor \hat{M}^2 \rfloor + k + 1$, if at most $2 - k$ of the next two slots are not non-tight.

To establish (46), a few more definitions are needed. Some definitions and lemmas that follow apply to a generic class, Class c, and are indicated as such. The remaining are specific to Class 2.

Holes [**8**]. Sometimes, the tasks in $\hat{\tau}^c$ may not be able to fully utilize the processors allocated to them. If h processors are idle at some time t, then we say that there are h *holes* (with respect to $\hat{\tau}^c$) in t. Thus, a tight (or pseudo-tight) slot has holes if fewer than $\lfloor \hat{M}^c \rfloor$ tasks of $\hat{\tau}^c$ are scheduled there, and a non-tight slot has holes if fewer than $\lceil \hat{M}^c \rceil$ tasks are scheduled there.

Task classification [8]. Tasks within class $\hat{\tau}^c$ may be classified as follows, based on whether they are active at t.[3]

$A(t)$**:** Set of all tasks that are active and scheduled at t.

$B(t)$**:** Set of all tasks that are active, but not scheduled at t.

$I(t)$**:** Set of all tasks that are inactive at t.

$A(t)$, $B(t)$, and $I(t)$ form a partition of $\hat{\tau}^c$, *i.e.*,

$$A(t) \cup B(t) \cup I(t) = \hat{\tau}^c \ \wedge$$
$$A(t) \cap B(t) = B(t) \cap I(t) = I(t) \cap A(t) = \emptyset. \quad (47)$$

The following lemma concerns set $B(t)$.

Lemma 36 *If t is a pseudo-tight slot and $LAG(\hat{\tau}^c, t+1) \geq LAG(\hat{\tau}^c, t)$, then $B(t)$ is not empty.*

Proof: Let $h \geq 0$ be the number of holes in t. Then, because t is a pseudo-tight slot, the number of tasks scheduled at t is $\lfloor \hat{M}^c \rfloor - h$. By (12), $LAG(\hat{\tau}^c, t+1) = LAG(\hat{\tau}^c, t) + \sum_{T \in \hat{\tau}^c} share(T,t) - (\lfloor \hat{M}^c \rfloor - h)$, which by (47) equals $LAG(\hat{\tau}^c, t) + \sum_{T \in A(t) \cup B(t) \cup I(t)} share(T,t) - (\lfloor \hat{M}^c \rfloor - h)$. Because $share(T,t) = 0$ for $T \in I(t)$, it follows that $LAG(\hat{\tau}^c, t+1) = LAG(\hat{\tau}^c, t) + \sum_{T \in A(t) \cup B(t)} share(T,t) - (\lfloor \hat{M}^c \rfloor - h)$. Because $LAG(\hat{\tau}^c, t+1) \geq LAG(\hat{\tau}^c, t)$ by the statement of the lemma, we have $\sum_{T \in A(t) \cup B(t)} share(T,t) \geq \lfloor \hat{M}^c \rfloor - h$. By (10), $share(T,t) \leq wt(T)$, for all T. Hence, $\sum_{T \in A(t) \cup B(t)} wt(T) \geq \lfloor \hat{M}^c \rfloor - h$. Because $|A(t)| = \lfloor \hat{M}^c \rfloor - h$ and $wt(T) < 1$ for all T, we have $\sum_{T \in A(t)} wt(T) < \lfloor \hat{M}^c \rfloor - h$. This, in turn, implies that $\sum_{T \in B(t)} wt(T) > 0$, and that $B(t)$ is not empty. □

The definition that follows identifies the last-released subtask at t of any task U.

Definition 5: Subtask U_j is the *critical subtask of U at t* iff $e(U_j) \leq t < d(U_j)$ and no other subtask U_k of U, where $k > j$, satisfies $e(U_k) \leq t < d(U_k)$.

Lemma 37 *If t is a pseudo-tight slot and $B(t)$ is not empty, then the critical subtask of every task in $B(t)$ is scheduled before t.*

Proof: Let U be any task in $B(t)$. Because U is active at t, t lies within the eligibility interval of some subtask of U. Let U_k be any subtask satisfying $e(U_k) \leq t < d(U_k)$ such that U_k is scheduled after t and its predecessor (if it exists) is scheduled before t. (Because $U \in B(t)$, no subtask of U is scheduled at t.) Because t is a pseudo-tight slot no tasks other than those scheduled at t are eligible at t and the number of such tasks is at most $\lfloor \hat{M}^c \rfloor$. Hence, if U_k were eligible at t, in addition to the $\lfloor \hat{M}^c \rfloor$ subtasks scheduled at t, then t would not be a pseudo-tight slot, which is a contradiction. Thus, for all subtasks U_k with $e(U_k) \leq t < d(U_k)$, U_k is scheduled before t, and in particular, this is true of U's critical subtask at t. □

Pseudo-tight slots are similar to non-tight slots with holes in the following sense: in a pseudo-tight slot, up to $\lceil \hat{M}^c \rceil$ processors are *potentially* available to the tasks in $\hat{\tau}^c$ but at most $\lfloor \hat{M}^c \rfloor$ such tasks are eligible; in a non-tight slot with holes, $\lceil \hat{M}^c \rceil$ are (definitely) available to the tasks in $\hat{\tau}^c$ but at most $\lfloor \hat{M}^c \rfloor$ such tasks are eligible. Due to this similarity, the following lemma, which was originally proved in [5] concerning slots with holes, is valid.

Lemma 38 [5] *Let t be a slot with holes or a pseudo-tight slot, and let $B(t)$ be non-empty. Let t_b be the latest time that a task in $B(t)$ is scheduled before t. Then, there exists a subtask W_l scheduled at t with $e(W_l) \leq t_b$, $d(W_l) = t+1$, and $S(W, t') = 0$, for all $t' \in [t_b, t-1]$. Also, there are no holes in $[t_b, t-1]$.*

The next four lemmas give estimates of lags of tasks in $A(t)$, $B(t)$, and $I(t)$ of $\hat{\tau}^2$ at time $t+1$, where t is a pseudo-tight slot, or a slot with holes. The first two of these are proved in [9], and apply to any $\hat{\tau}^c$. The third can be proved similarly.

Lemma 39 [9] *Let t be a slot with holes or a pseudo-tight slot in the schedule S^c for $\hat{\tau}^c$, and let $W \in I(t)$. Then $lag(W, t+1) = 0$.*

Lemma 40 [9] *Let t be a slot with holes or a pseudo-tight slot in the schedule S^c for $\hat{\tau}^c$, and let $W \in B(t)$. Then $lag(W, t+1) \leq 0$.*

Lemma 41 *Let $t < t_d$ be a slot with holes or a pseudo-tight in the schedule S^2 for $\hat{\tau}^2$, slot and let $W \in A(t)$. Then $lag(W, t+1) < 3 \cdot wt(W)$.*

Lemma 42 *Let $t < t_d$ be a slot with holes or a pseudo-tight slot in the schedule S^c for $\hat{\tau}^c$, and let $B(t)$ be non-empty. Then, there exists a task $W \in A(t)$ such that $lag(W, t+1) < wt(W)$.*

Proof: By Lemma 38, a subtask W_l is scheduled at t with $d(W_l) = t+1$. Let W_k denote W_l's successor. (If no successor exists, the reasoning is as in Case 1 below.) We consider two cases depending on the value of $r(W_k)$.

Case 1: $r(W_k) \geq t+1$. In this case, both the actual and ideal systems completely execute all subtasks of W through W_l by time $t+1$ and no later subtasks of W. Therefore, $lag(W, t+1) = 0$.

[3] For brevity, we let the task system $\hat{\tau}^c$ to be implicit in these definitions.

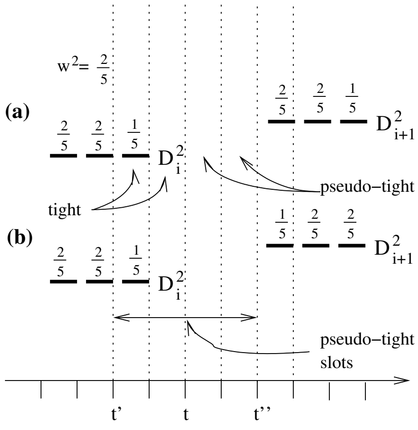


Figure 11. Lemma 44.

Case 2: $r(W_k) \geq t$**.** Note that, in this case, the windows of W_l and W_k overlap by one slot, and hence by the definition of a GIS system, $k = l+1$. By (3)–(4), we have $r(W_{l+2}) \geq t+1$. Therefore, any additional share that W may receive by time t in the ideal system is due to the flow received by W_{l+1} in the first slot of its window, *i.e.*, $f(W_{l+1}, t)$. Using (10), it can be shown that $f(W_{l+1}, t) < wt(W)$. Hence, $lag(W, t+1) < wt(W)$. ☐

Lemma 43 $t_d - 1$ *is not pseudo-tight.*

Proof: The proof is by contradiction. By Lemma 31, at least $2 \cdot \lfloor \hat{M}^2 \rfloor + k$, where $1 \leq k \leq 3$, is the number of subtasks of $\hat{\tau}^2$ with deadlines at or prior to t_d that are not scheduled by t_d, and at most $k-1$ of the next two slots, t_d and t_d+1 are non-tight. Because $k \geq 1$, at least $2 \cdot \lfloor \hat{M}^2 \rfloor + 1$ subtasks miss their deadlines at t_d, and because $t_d - 1$ is pseudo-tight, the number of tasks scheduled at $t_d - 1$ is at most $\lfloor \hat{M}^2 \rfloor$. That $t_d - 1$ is pseudo-tight implies that at most $\lfloor \hat{M}^2 \rfloor$ tasks are eligible at $t_d - 1$, and that all the subtasks missing their deadlines at t_d have their predecessors scheduled at $t_d - 1$. That would imply that at least one task has three of its subtasks missing their deadlines at t_d, which by (3) implies that the tardiness of at least one subtask with a deadline earlier than $t_d - 1$ and scheduled at $t_d - 1$ is at least three, contradicting (A2). ☐

The three lemmas that follow will be used to show that (46) is true.

Lemma 44 *Let* $t < t_d$ *be a pseudo-tight slot,* $t'' > t$*, the earliest time slot after* t *that* D^2 *is active, and* $t' < t''$*, the latest time at or before* t *that* D^2 *was active. If* $share(D^2, t'') < w^2$ *and* $LAG(\hat{\tau}^2, t') \leq 2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t') + w^2$*, then* $LAG(\hat{\tau}^2, t+1) \leq 2 \cdot \lfloor \hat{M}^2 \rfloor - share(D^2, t') + w^2$*. Else,* $LAG(\hat{\tau}^2, t+1) < 2 \cdot \lfloor \hat{M}^2 \rfloor$*, which is less than* $2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t+1) + w^2$*.*

Proof Sketch: The conditions of the lemma are illustrated in Fig. 11. We first show that $LAG(\hat{\tau}^2, t+1) < 2 \cdot \lfloor \hat{M}^2 \rfloor$, regardless of D^2's share in t''. By (47), $LAG(\hat{\tau}^2, t+1) = \sum_{T \in A(t) \cup B(t) \cup I(t)} lag(T, t+1)$, which by Lemmas 39–41 implies that $LAG(\hat{\tau}^2, t+1) < \sum_{T \in A(t)} 3 \cdot wt(T)$. Because t is pseudo-tight, $|A(t)| \leq \lfloor \hat{M}^2 \rfloor$, which by (L2) yields $LAG(\hat{\tau}^2, t+1) < 2 \cdot \lfloor \hat{M}^2 \rfloor$. By Lemma 30, $lag(D^2, t+1)$ is zero, and hence, $LAG(\hat{\tau}^2, t+1)$ is less than $2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t+1) + w^2$.

Next, consider the case that the share of D^2 in t'' in the ideal system is strictly less than its weight. This is illustrated in Fig. 11(b). Let D^2_{i+1} be D^2's subtask that is released at t''. If t' is the latest time slot at or before t that D^2 was active, then $t'+1$ should be the deadline of the predecessor of D^2_{i+1}. By Lemma 29, D^2_{i+1}'s predecessor is D^2_i, and hence $t' = d(D^2_i) - 1$. Therefore, by Lemma 29, we also have

$$(\forall t : t \in [t', t''-1] :: t \text{ is pseudo-tight}). \tag{48}$$

We next show that the bound stated in this lemma ($2 \cdot \lfloor \hat{M}^2 \rfloor - share(D^2, t') + w^2$) holds at every t in $L = [t', t'']$, by induction on t. For the base case, let $t = t'$. Because t' is pseudo-tight by (48) and D^2 is active in t', by Lemma 27,

$$lag(D^2, t') = -share(D^2, t'). \tag{49}$$

From the statement of the lemma, $LAG(\hat{\tau}^2, t') \leq 2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t') + w^2$, which by the above equation implies that $LAG(\hat{\tau}^2, t') \leq 2 \cdot \lfloor \hat{M}^2 \rfloor - share(D^2, t') + w^2$. This establishes the base case.

Next, assuming that LAG is at most $2 \cdot \lfloor \hat{M}^2 \rfloor - share(D^2, t') + w^2$, at t' through t, where $t < t''$, we show that the same bound applies at $t+1$. We consider two cases depending on whether $B(t)$ is empty or not.

Case 1: $B(t) = \emptyset$**.** By (12) and (47), $LAG(\hat{\tau}^2, t+1) = LAG(\hat{\tau}^2, t) + \sum_{T \in A(t)} share(T, t) - \sum_{T \in A(t)} S(T, t)$. Because the share of every task is at most its weight, which is strictly less than one, and the number of tasks scheduled at t equals $A(t)$, the last equation above implies that $LAG(\hat{\tau}^2, t+1) < LAG(\hat{\tau}^2, t)$. Therefore, by our induction hypothesis, we have $LAG(\hat{\tau}^2, t+1) < 2 \cdot \lfloor \hat{M}^2 \rfloor - share(D^2, t') + w^2$.

Case 2: $B(t) \neq \emptyset$**.** For this case, by (47) and Lemmas 39–41, we have $LAG(\hat{\tau}^2, t+1) \leq \sum_{T \in A(t)} lag(T, t+1)$. By Lemmas 41 and 42, the lag of at least one task W in $A(t)$ is less than its weight, and the lag of the remaining tasks, less than thrice their weights. Therefore, $LAG(\hat{\tau}^2, t+1)$ is less than $wt(W) + \sum_{T \in A(t) - \{W\}} 3 \cdot wt(T)$. t is pseudo-tight by (48), and hence, $|A(t)| \leq \lfloor \hat{M}^2 \rfloor$. This, together with (L2), implies that $LAG(\hat{\tau}^2, t+1) < 2 \cdot \lfloor \hat{M}^2 \rfloor - 1$, which is less than $2 \cdot \lfloor \hat{M}^2 \rfloor - share(D^2, t') + w^2$ (because

$share(D^2, t') < 1$). $\square$

Lemma 45 *Let $t < t_d$ be a tight slot and let $LAG(\hat{\tau}^2, t) \leq 2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t) + w^2$. Then, there exists a t', such that $t < t' \leq t_d$ and $LAG(\hat{\tau}^2, t') \leq 2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t') + w^2$.*

Proof: Based on whether there are holes in t or not, we consider the following two cases.

Case 1: t contains one or more holes. Let $h > 0$ be the number of holes in t. Then, $|A(t)| = \lfloor \hat{M}^2 \rfloor - h$. By (47), $LAG(\hat{\tau}^2, t+1) = \sum_{T \in A(t) \cup B(t) \cup I(t)} lag(T, t+1)$, which is then at most $2 \cdot \lfloor \hat{M}^2 \rfloor - 2h$, by Lemmas 39–41 and (L2). Because $h \geq 1$, $2 \cdot \lfloor \hat{M}^2 \rfloor - 2h \leq 2 \cdot \lfloor \hat{M}^2 \rfloor - 2$, which is less than $2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t+1) + w^2$, by Lemma 33. Thus, $t' = t + 1$ for this case.

Case 2: There are no holes in t. Because t is tight, $S(D^2, t) = 0$. Therefore, by (11),

$$lag(D^2, t+1) = lag(D^2, t) + share(D^2, t). \qquad (50)$$

Because there are no holes in t, $\sum_{T \in \hat{\tau}^2} S(T, t) = \lfloor \hat{M}^2 \rfloor$. Therefore, by (12),

$$\begin{aligned} LAG(\hat{\tau}^2, t+1) &\leq LAG(\hat{\tau}^2, t) + \hat{M}^2 - \lfloor \hat{M}^2 \rfloor \\ &= LAG(\hat{\tau}^2, t) + w^2 \qquad \text{(by Lemma 25)} \qquad (51) \\ &\leq 2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t) + 2 \cdot w^2. \qquad (52) \\ &\text{(By the statement of the lemma.)} \end{aligned}$$

We consider two subcases, based on the share that D^2 receives in the ideal system.

Subcase 2(a): $share(D^2, t) = w^2$. For this case, by (50), $lag(D^2, t+1) = lag(D^2, t) + w^2$, which can be substituted in (52) to yield $LAG(\hat{\tau}^2, t+1) \leq 2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t+1) + w^2$. Thus, $t' = t + 1$ for this subcase, too.

Subcase 2(b): $0 < share(D^2, t) < w^2$. By Lemma 28, the condition of this subcase implies that either $t = r(D_i^2)$ or $t = d(D_i^2) - 1$. We consider both possibilities.

If $t = r(D_i^2)$, then by Lemma 28, D_i^2 is released late. From line 12 of I-EPDF, the postponement of release time is by at most one time unit. Therefore, D_i^2's release should have been postponed at $t - 1$, *i.e.*,

$$t - 1 \text{ is pseudo-tight.} \qquad (53)$$

Hence, by Lemma 44, $LAG(\hat{\tau}^2, t) \leq 2 \cdot \lfloor \hat{M}^2 \rfloor - share(D^2, t') + w^2$, where t' is the latest time slot before t that D^2 was active. Therefore, $t' + 1$ should be the deadline of D^2i's predecessor. With the help of Lemma 29, it can be concluded that D_{i-1}^2 is D_i^2's predecessor in S^1, $t' = d(D_{i-1}^2) - 1$, and $share(D^2, t') = 1 - share(D^2, t)$. Substituting this value in the previous expression for LAG at t, we have $LAG(\hat{\tau}^2, t) \leq 2 \cdot \lfloor \hat{M}^2 \rfloor - 1 + share(D^2, t) + w^2$, which on substitution in (51) gives

$$\begin{aligned} &LAG(\hat{\tau}^2, t+1) \\ &\leq 2 \cdot \lfloor \hat{M}^2 \rfloor - 1 + share(D^2, t) + 2 \cdot w^2 \\ &= 2 \cdot \lfloor \hat{M}^2 \rfloor - 1 + lag(D^2, t+1) - lag(D^2, t) + 2 \cdot w^2 \\ &\qquad \text{(from 50)} \\ &= 2 \cdot \lfloor \hat{M}^2 \rfloor - 1 + lag(D^2, t+1) + 2 \cdot w^2 \\ &\qquad \text{(By (53) and Lemma 30, } lag(D^2, t) = 0) \\ &< 2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t+1) + w^2. \qquad \text{(by (L2))} \end{aligned}$$

On the other hand, if $t = d(D_i^2) - 1$, then by Lemma 28, D_{i+1}^2 is released late. Let t' be its eventual time of release. Then $t' - 1$ is pseudo-tight, and hence by Lemma 44, $LAG(\hat{\tau}^2, t')$ is less than $2 \cdot \lfloor \hat{M}^2 \rfloor + w^2$. Because $t' - 1$ is pseudo-tight, $lag(D^2, t') = 0$, by Lemma 30. Therefore, $LAG(\hat{\tau}^2, t') < 2 \cdot \lfloor \hat{M}^2 \rfloor + w^2 \leq 2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t') + w^2$. By Lemma 43, $t_d - 1$ is not pseudo-tight. Therefore, $t' \leq t_d - 1$. $\square$

Lemma 46 *Let $t < t_d$ be a non-tight slot and let $LAG(\hat{\tau}^2, t) \leq 2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t) + w^2$. Then, either $LAG(\hat{\tau}^2, t+1) \leq 2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t+1) + w^2$ or $LAG(\hat{\tau}^2, t_d) < 2 \cdot \lfloor \hat{M}^2 \rfloor + 1$ holds.*

Proof: As with Lemma 45, we broadly consider two cases depending on whether there are holes in t or not.

Case 1: t contains one or more holes. By (47) and Lemmas 39–40, $LAG(\hat{\tau}^2, t+1) < \sum_{T \in A(t)} 3 \cdot wt(T)$. Because t contains holes, $|A(t)| \leq \lfloor \hat{M}^2 \rfloor$, which together with (L2) implies that

$$LAG(\hat{\tau}^2, t+1) < 2 \cdot \lfloor \hat{M}^2 \rfloor. \qquad (54)$$

Because t is non-tight, it can contain a hole only under the conditions in either of the two subcases that follow.

Subcase 1(a): $t = d(D_i^2) - 1$ or $t = d(D_i^2)$, for some i, and D_i^2 is scheduled at t in S^1. That it is possible for t to contain a hole can be verified from lines 5–13 in Fig. 5. Because $t + 1 \geq d(D_i^2)$ and S^1 does not execute any other subtask released later than D_i^2 before $t + 1$, it follows that

$$lag(D^2, t+1) \geq 0. \qquad (55)$$

On the other hand, by (54), we have $LAG(\hat{\tau}^2, t+1) < 2 \cdot \lfloor \hat{M}^2 \rfloor$, where, by (55), $2 \cdot \lfloor \hat{M}^2 \rfloor < 2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, t+1) + w^2$. This completes the proof for this subcase.

Subcase 1(b): $r(D_i^2) \leq t < d(D_i^2) - 1$, D_i^2 is scheduled at t in S^1, the number of subtasks of $\hat{\tau}^2$ that are eligible and scheduled at t is $\lfloor \hat{M}^2 \rfloor + 1 - n$, where $n \geq 1$, and there exists a set of subtasks τ', such that $|\tau'| \geq n$ and the following

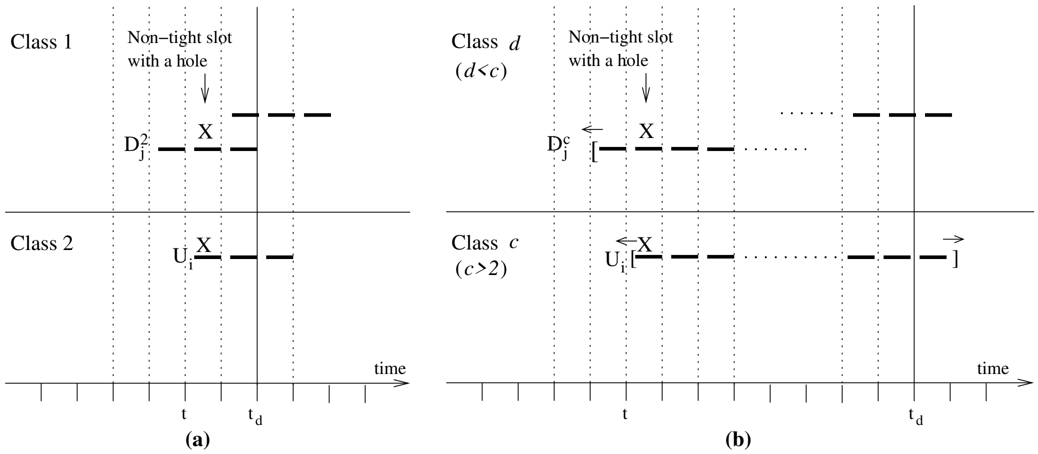


Figure 12. Lemma 46**(a)** Subcase 1(b)-i. $wt(U) \geq 1/2$. Because $d(U_i) > t_d$, U_i is not present in S^2. Therefore, t is a non-tight slot with holes for $\hat{\tau}^2$. **(b)** Subcase 1(b)-ii. $wt(U) < 1/2$.

holds.

$$(\forall T_i \in \tau' :: r(T_i) \leq t \wedge\ d(T_i) > t_d\ \wedge \text{ all subtasks of } T \text{ preceding } T_i \text{ are scheduled in } S^2 \text{ before } t) \quad (56)$$

Then, the tasks that are eligible at t consist of the tasks of subtasks of $\hat{\tau}^2$ that are scheduled at t and the tasks of subtasks in τ'. Therefore, $\mathcal{E}_2$, as determined at t in line 13 of Fig. 5 is at least $\lfloor \hat{M}^2 \rfloor + 1$ (or $P_2 + 1$). Therefore, the test in line 8 fails, and the release time of D_i^2 is not postponed. However, the subtasks in τ' are not in $\hat{\tau}^2$, by (A3) and (56). Thus, the removal of the n subtasks of τ' from the actual schedule to yield S^2, results in one or more holes in t in S^2. This is illustrated in Fig. 12. We show that $LAG(\hat{\tau}^2, t_d) < 2 \cdot \lfloor \hat{M}^2 \rfloor + 1$ as follows. Without loss of generality, we assume that t is the latest slot with a hole and that there is no pseudo-tight slot after t. (Otherwise, it suffices to consider a later slot with a hole or a later pseudo-tight slot.)

Let U be the task of some subtask in τ'. Then, by (L2), the weight of U is greater than $1/2$. However, if $c > 2$ and Class c is of Type 3, then it is possible that $wt(U) < 1/2$. Therefore, we consider the two subcases that follow.

Subcase 1(b)-i: $c = 2$ **and** $wt(U) \geq 1/2$**.** Fig. 12(a) illustrates this subcase. Let U_i be a subtask in τ'. Because $wt(U) \geq 1/2$, by Lemma 1, U_i spans at most three slots, *i.e.* $d(U_i) - r(U_i) \leq 3$, which by (56) implies that $t_d \leq t + 2$. If $t_d = t + 1$, then (54) establishes the lemma. On the other hand, if $t_d = t + 2$, then as explained earlier, $t + 1$ neither contains holes nor is it a pseudo-tight slot. Therefore, if $t + 1$ is a non-tight slot, then by (12), $LAG(\hat{\tau}^2, t + 2) = LAG(\hat{\tau}^2, t + 1) + \hat{M}^2 - (\lfloor \hat{M}^2 \rfloor + 1)$, which is less than $2 \cdot \lfloor \hat{M}^2 \rfloor$, by (54), and if $t + 1$ is a tight slot, then $LAG(\hat{\tau}^2, t+2) = LAG(\hat{\tau}^2, t+1) + \hat{M}^2 - \lfloor \hat{M}^2 \rfloor$, which by (54) and Lemma 25 is less than $2 \cdot \lfloor \hat{M}^2 \rfloor + w^2$, which is less than $2 \cdot \lfloor \hat{M}^2 \rfloor + 1$.

Subcase 1(b)-ii: $c > 2$ **and** $wt(U) < 1/2$**.** As explained in the next subsection, (46) can be generalized to $LAG(\hat{\tau}^c, t_d) \leq c \cdot \lfloor \hat{M}^c \rfloor + lag(D^c, t_d) + w^c$ for Class c. Because $wt(U) < 1/2$ for some task in $\hat{\tau}^c$, where $c > 2$, (W) implies that U is a donor task, say D^m, added to Class c. Therefore, by Lemma 23, we have

$$wt(U) > w^c. \quad (57)$$

Let U_i be the subtask of U that is scheduled at t in the non-truncated schedule for Class c and let D_j^c be the subtask of D^c that is scheduled at t in S^d, the schedule for Class d, where $d < c$ is the class that Class c borrows from. This is illustrated in Fig. 12(b). Then, by (57) and Lemma 1, $|w(U_i)| \leq \left\lceil \frac{1}{w^c} \right\rceil + 1$. Because $U_i \in \tau'$, by (56) we have $d(U_i) > t_d$, $r(U_i) \leq t$, and hence, $|w(U_i)| = d(U_i) - r(U_i) \geq t_d + 1 - t$, which implies that $t_d - (t + 1) \leq |w(U_i)| - 2 \leq \left\lceil \frac{1}{w^c} \right\rceil - 1$. Generalizing (54), we have $LAG(\hat{\tau}^c, t + 1) < c \cdot \lfloor \hat{M}^c \rfloor$. Because there are no slots with holes or pseudo-tight slots after t, it can be easily shown that the total LAG of $\hat{\tau}^c$ increases by at most w^c across any slot in $[t + 1, t_d]$. Therefore, $LAG(\hat{\tau}^c, t_d) \leq LAG(\hat{\tau}^c, t + 1) + (\left\lceil \frac{1}{w^c} \right\rceil - 1) \cdot w^c < c \cdot \lfloor \hat{M}^c \rfloor + 1$.

Case 2: There are no holes in t. This case can be proved in a manner that is exactly similar to the proof for Case 2 of Lemma 45. □

Because $LAG(\hat{\tau}^2, 0) = 0$ and $lag(D^2, 0) = 0$, we have $LAG(\hat{\tau}^2, 0) < 2 \cdot \lfloor \hat{M}^2 \rfloor + lag(D^2, 0) + w^2$. Therefore, by Lemmas 44–46 above, either (46) is true, or $LAG(\hat{\tau}^2, t_d) <$

$2 \cdot \lfloor \hat{M}^2 \rfloor + 1$, which contradict Lemma 32. Therefore, (A3) is false, or the tardiness of $\hat{\tau}^2$ under I-EPDF is at most two.

C.3 Tardiness Bounds for $\hat{\tau}^3$

Finally, we are left with showing that the tardiness bounds of classes of Type 3 are met under I-EPDF.

We begin by considering the following difference of interest between a Type 2 class and a Type 3 class, and its impact on the proof presented for a Type 2 class.

> The weight of the donor tasks added to a Type 3 class may exceed the weight restrictions of the class, *i.e.*, the weight of a donor task D^m added to Class c may be greater than $\frac{c}{c+1}$. However, by Lemma 24 and (W), the number of such tasks is at most one.

In this subsection, we consider in detail the implications of this difference — the scheduling rule that it necessitates and the changes to the proof presented in the previous section that it entails.

The principal hurdle imposed when the weight of a donor task (or any task for that matter) of a class exceeds the maximum permissible for that class is the following. The proof strategy of the previous section consists of bounding the lag of a task system at the end of a pseudo-tight slot or a slot with holes by the sum of the lags of the tasks scheduled in that slot. Because the lags of individual tasks are determined by their weights, the existence of tasks that violate the weight restrictions results in the bound determined using this approach to exceed the value that needs be established for (46) to hold. We overcome this hurdle as follows.

We first observe that because the donor task of Class 3 (D^3) is added to Class 1, $tardiness(D^3) = 1$, which is two less than the tardiness that needs to be ensured for tasks in Class 3. As proved later, this property can be exploited to weaken the right-hand side in the counterpart of (46) for Class 3 to $3 \cdot \lfloor \hat{M}^3 \rfloor + lag(D^3, t_d) + 2 \cdot w^3$. Note that the difference is in the last term. By Lemma 22, if the weight of the donor task D^m added to Class 3 is at least $5/6$, then D^3 is heavy. This implies that $2 \cdot w^3 \geq 1$, which is sufficient to counter the increase in the total lag due to the presence of an extra-heavy donor task. On the other hand, if $\frac{c}{c+1} < w^m \leq 5/6$, we show that the tardiness bounds of any Class c are not violated, even if the tardiness of D^c is as high as $c - 1$. (For Class 3 it would mean that $tardiness(D^3)$ can be up to two.)

However, the difference between the tardiness that needs to be ensured for a Type 3 class, say Class c, and the tardiness of its donor task D^c may not be two, if D^c is not added to Class 1. To ensure this difference for all classes higher than three, we introduce the following scheduling rule.

(R): Break ties involving the heavy donor task, if any, of Class 3 and higher, in favor of the donor task.

As shown later, (R) ensures a tardiness of at most $k - 1$ for the heavy donor task of Class k, if (46) is slightly strengthened. (If Class 3 or above is of Type 1, then it can easily be shown that no donor task added to the class is heavy. Hence, (R) will not be applicable to that class, and therefore, the argument given for Type 1 classes is not impacted.)

The rest of this section contains a formal proof of the informal summary given above.

Our goal now is to show the following.

(G1) I-EPDF ensures a tardiness of at most c for Class c that is of type 3 and is augmented with a donor task D^m with weight w^m exceeding $\frac{c}{c+1}$.

(G2) I-EPDF (specifically, (R) of I-EPDF) ensures a tardiness of at most $c-1$ for a heavy donor task added to a Type 3 class, c.

Proof of (G1) is exactly similar to that of a Type 2 class presented in Sec. C.2, if (G2) is known to hold. The justification is as follows. Generalizing the arguments of Sec. C.2, our goal would be to show that $LAG(\hat{\tau}^c, t) < c \cdot \lfloor \hat{M}^c \rfloor + k$, where k ranges between one and $c + 1$, depending on the tightness of the c slots starting at t. If (G2) holds, then by Lemma 53, the lag of D^m at the end of a slot with holes is at most c. By the same lemma, it can easily be shown that the lag of a task in τ^c at the end of such slots is also at most c. Therefore, D^m does not cause the total lag of $\hat{\tau}^c$ to be more than what it would be if D^m were like any other class of $\hat{\tau}^c$, *i.e.*, $w^c \leq \frac{c}{c+1}$ and tardiness of D^m is c. Hence, the proof steps of the previous subsection can be used to show that tardiness for Class c is at most c. Therefore, it suffices to show that (G2) holds.

We prove (G2) for two distinct cases.

Case A: $5/6 \leq w^m < 1$.

Case B: $1/2 \leq w^m < 5/6$.

C.3.1 Case A: $5/6 \leq w^m < 1$

As mentioned earlier, for simplicity, we prove this case for Class 3, *i.e.*, $c = 3$, and as before, the proof is by contradiction.

If (G2) does not hold, then t_d, τ, and $\hat{\tau}^3$ defined as follows exist.

Definition 6: t_d is the earliest of the deadlines of the subtasks of D^m with a tardiness of three under I-EPDF, *i.e.*, there exists some task system τ with a subtask D_l^m of D^m in $\hat{\tau}^3$ with a deadline at t_d and a tardiness of three, and there does not exist any other task system σ with a heavy donor task D^k in $\hat{\sigma}^3$ with a subtask with a deadline prior to t_d and a tardiness of three. Further, the tardiness of all other tasks in $\hat{\tau}^3$ is at most three at t_d.

Definition 7: τ is a feasible task system with the following properties.

(T1) t_d is the earliest deadline of a subtask of D^m in $\hat{\tau}^3$ with a tardiness of three under I-EPDF. The tardiness of all other tasks of $\hat{\tau}^3$ is at most three at t_d.

(T2) No other feasible task system σ satisfying (T1) contains fewer subtasks of tasks in $\hat{\sigma}^3$ than τ.

(T3) No feasible task system σ satisfying (T1) and (T2) has a larger rank than τ, where the *rank* of a task system τ at t is defined as $\sum_{T \in \hat{\tau}^3} \sum_{\{T_i | d(T_i) \le t_d\}} e(T_i)$.

By (T1) and (T2), the deadline of every subtask in $\hat{\tau}^3$ other than D_l^m is at most $t_d - 1$. Since D^m has a higher priority over other tasks if there are ties, the presence of a subtask with a deadline at or after t_d cannot affect where D_l^m is scheduled, and hence can be removed. Additionally, the following assertions follow from the above properties and definitions.

(B1) $\exists D_l^m \in \hat{\tau}^3 : d(D_l^m) = t_d \wedge \ tardiness(D_l^m) = 3$

(B2) $(\forall T_i \in \hat{\tau}^3 : ((T \ne D^m \Rightarrow d(T_i) < t_d \wedge \ tardiness(T_i) \le 3) \wedge ((T = D^m \wedge \ d(T_i) < t_d) \Rightarrow tardiness(T_i) \le 2)))$

We first show that the number of subtasks of $\hat{\tau}^3$ that miss their deadlines at $t_d - 1$ is at least $3 \cdot \lfloor \hat{M}^3 \rfloor$.

Lemma 47 *The number of subtasks that have their deadlines at or prior to* $t_d - 1$ *and are not scheduled by* $t_d - 1$ *is at least* $3 \cdot \lfloor \hat{M}^3 \rfloor + k$*, where* k *is the number of slots in* $[t_d - 1, t_d + 1]$ *that are non-tight.*

Proof: We prove the lemma for $k = 0$; the other cases are similar. By (B1), D_l^m is scheduled at $t_d + 2$. Because all of $t_d - 1$, t_d, and $t_d + 1$ are tight, only $\lfloor \hat{M}^3 \rfloor$ processors are available to the tasks in $\hat{\tau}^3$ in each of these slots. By (B2), it suffices to show that $\lfloor \hat{M}^3 \rfloor$ subtasks from $\hat{\tau}^3$ are scheduled in each of these slots. First, suppose that fewer than $\lfloor \hat{M}^3 \rfloor$ such subtasks are scheduled in slot $t_d + 1$. Then, because D_l^m could not be scheduled there, its predecessor must have been scheduled at $t_d + 1$. But then, D_l^m's predecessor would have a tardiness of at least 3, contradicting the fact that D_l^m is the only subtask of D^m with a tardiness of three. Having shown that $\lfloor \hat{M}^3 \rfloor$ subtasks from $\hat{\tau}^3$ are scheduled at $t_d + 1$, it follows that the same is true of t_d and $t_d - 1$. Otherwise, one of the subtasks scheduled at $t_d + 1$ (t_d) would have been eligible at t_d ($t_d - 1$) and hence, would have been scheduled there. □

From the above lemma, we have the following.

Lemma 48 $LAG(\hat{\tau}^3, t_d - 1)$ *is at least* $3 \cdot \lfloor \hat{M}^3 \rfloor + k$*, where* k *is the number of slots in* $[t_d - 1, t_d + 1]$ *that are non-tight.*

Proof: Similar to the proof of Lemma 32. □

We next present lemmas concerning lag of D^3 at t, which will be used to derive an expression for the total lag of $\hat{\tau}^3$. Because D^3 is in $\hat{\tau}^1$, the proofs of the first two lemmas are the same as the proofs of Lemmas 33 and 34, respectively.

Lemma 49 $-1 < lag(D^3, t) < 1 + w^3$*, for all* t*.*

Lemma 50 *If* $lag(D^3, t) \ge 1$*, and the next releasable subtask of* D^3 *is not postponed at* t*, then* t *is non-tight.*

The lemma that follows is the counterpart of Lemma 35 for D^3.

Lemma 51 *If* $lag(D^3, t) \ge k - 2 \cdot w^3$*, where* $1 \le k \le 3$*, and the next releasable subtask of* D^3 *is not postponed at* t*,* $t + 1$*, and* $t + 2$*, then at least* k *of the three slots in* $[t, t + 2]$ *are non-tight.*

Proof: We prove the lemma for $k = 1$. If t or $t + 1$ is non-tight, then the proof is complete. Therefore, assume otherwise. If $lag(D^3, t) \ge 1 - 2 \cdot w^3$, then by (11), $lag(D^2, t + 2) \ge 1$, because there are no IS separations and D^3 is not scheduled at t and $t + 1$. The required result then follows from Lemma 50. □

In what follows, we obtain a contradiction to Lemma 48 by showing that $LAG(\hat{\tau}^3, t_d - 1)$ is less than $3 \cdot \lfloor \hat{M}^3 \rfloor + k$, where k of the three slots starting at $t_d - 1$ are non-tight. We do this by showing that

$$LAG(\hat{\tau}^3, t_d - 1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t_d - 1) + 2 \cdot w^3, \tag{58}$$

and appealing to Lemmas 49–51 to imply the contradiction.

As mentioned in the previous subsection, several lemmas proved in that subsection apply to any generic class Class c, and hence to Class 3. We borrow them where needed.

Lemma 52 *Let* $t < t_d - 1$ *be a slot with holes or a pseudo-tight slot in schedule* S^3 *for* $\hat{\tau}^3$*, and let* $W \in A(t)$*. Then* $lag(W, t + 1) < 4 \cdot wt(W)$*.*

Proof: Similar to the proof of Lemmas 39–41 of the previous section.

Lemma 53 *Let* t *be a slot with holes or a pseudo-tight slot in schedule* S^c *for* $\hat{\tau}^c$*, and let* $W \in A(t)$*. If the tardiness of the subtask of* W *scheduled at* t *is* $k \ge 0$*, then* $lag(W, t + 1) < (k + 1) \cdot wt(W)$*.*

Proof: Similar to the proof of Lemma 41.

Lemma 54 *Let* $t < t_d - 1$ *be a slot with holes or a pseudo-tight slot in schedule* S^3 *for* $\hat{\tau}^3$*, and let* $W \in A(t)$*. Then* $lag(W, t + 1) < 3$*.*

Proof: If $W \ne D^m$, then by (W), $4 \cdot wt(W) = 3$. Therefore, by Lemma 52, $lag(W, t+1) < 3$. On the other hand, if $W = D^m$, then $tardiness(W) = 2$. Therefore, by Lemma 25 and (14), $3 \cdot w^m < 3$, and by Lemma 53, $lag(W, t + 1) < 3$. □

Lemma 55 *Let $t < t_d - 1$ be a pseudo-tight slot, $t'' > t$, the earliest time slot after t that D^3 is active, and t', the latest time at or before t that D^3 was active. If $share(D^3, t'') < w^3$ and $LAG(\hat{\tau}^3, t') \leq 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t') + 2 \cdot w^3$, then $LAG(\hat{\tau}^3, t+1) \leq 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 - share(D^3, t') + 2 \cdot w^3$. Else, $LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor$, which is less than $3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t+1) + 2 \cdot w^3$.*

Proof Sketch: The conditions of the lemma are illustrated in Fig. 11 (Substitute D^3, for D^2, though). The proof of this lemma follows that of Lemma 44 closely. The differences are in the constants in the expression for LAG, and the bound on the lag of D^3. The proof for the first part is exactly similar to the proof of the corresponding part in Lemma 44, and hence it is omitted.

We next show that $LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor$, regardless of D^3's share in t''. By (47), $LAG(\hat{\tau}^3, t+1) = \sum_{T \in A(t) \cup B(t) \cup I(t)} lag(T, t+1)$. By Lemmas 40, 39, and 54, $LAG(\hat{\tau}^3, t+1) < \sum_{T \in A(t)} 3$. Because t is pseudo-tight, $|A(t)| \leq \lfloor \hat{M}^3 \rfloor$, and hence, $LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor$. Finally, we show that $3 \cdot \lfloor \hat{M}^3 \rfloor < 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t+1) + 2 \cdot w^3$. Because t is pseudo-tight, by Lemma 30, $lag(D^3, t+1) = 0$, and by the assumption for this case that $w^3 \geq 5/6$ and Lemma 22, $w^3 > 1/2$. Therefore, the required result follows. □

Lemma 56 *$t_d - 2$ is neither pseudo-tight nor does it contain holes.*

Proof: Similar to the proof of Lemma 43. □

Lemma 57 *Let $t < t_d - 1$ be a tight slot and let $LAG(\hat{\tau}^3, t) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t) + 2 \cdot w^3$. Then, there exists a t', such that $t < t' \leq t_d - 1$ and $LAG(\hat{\tau}^3, t') < 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t') + 2 \cdot w^3$.*

Proof: The proof is exactly similar to that of Lemma 45, and is hence, omitted. □

Lemma 58 *Let t be a non-tight slot and let $LAG(\hat{\tau}^3, t) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t) + 2 \cdot w^3$. Then, (i) $LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t+1) + 2 \cdot w^3$, or (ii) either $LAG(\hat{\tau}^3, t_d - 1) < 3 \cdot \lfloor \hat{M}^3 \rfloor$, or $LAG(\hat{\tau}^3, t_d - 1) < 3 \cdot \lfloor \hat{M}^3 \rfloor + 1$ and one of the slots in $[t_d - 1, t_d + 1]$ is non-tight.*

Proof: We broadly consider two cases depending on whether there are holes in t or not.

Case 1: t contains one or more holes. By (47) and Lemmas 39, 40, and 54, $LAG(\hat{\tau}^3, t+1) < \sum_{T \in A(t)} 3$. Because t contains holes, $|A(t)| \leq \lfloor \hat{M}^3 \rfloor$, which implies that

$$LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor. \tag{59}$$

Because t is non-tight, it can contain a hole only under the conditions in either of the two subcases that follow.

Subcase 1(a): $t = d(D_i^3) - 1$ or $t = d(D_i^3)$, for some i, and D_i^3 is scheduled at t in S^1. The proof for this subcase is similar to that of its counterpart in Lemma 46.

Subcase 1(b): $r(D_i^3) \leq t < d(D_i^3) - 1$, D_i^3 is scheduled at t in S^1, the number of subtasks of $\hat{\tau}^3$ that are eligible and scheduled at t is $\lfloor \hat{M}^3 \rfloor + 1 - n$, where $n \geq 1$, and there exists a set of subtasks τ', such that $|\tau'| \geq n$ and the following holds.

$$(\forall T_i \in \tau' :: r(T_i) \leq t \land d(T_i) > t_d - 1 \land \text{all subtasks of } T \text{ preceding } T_i \text{ are scheduled in } S^2 \text{ before } t) \tag{60}$$

Then, the tasks that are eligible at t consist of the tasks of subtasks of $\hat{\tau}^3$ that are scheduled at t in S^3 and the tasks of subtasks in τ'. Therefore, $\mathcal{E}_3$, as determined at t in line 13 of Fig. 5 is at least $\lfloor \hat{M}^3 \rfloor + 1$ (or $P_3 + 1$). Therefore, the test in line 8 fails, and the release time of D_i^3 is not postponed. However, the subtasks in τ' are not in $\hat{\tau}^3$, by (B2) and (60). Thus, the removal of the n subtasks of τ' from the actual schedule to yield S^3, results in one or more holes in t in S^3. We show that in this case expression (ii) in the statement of the lemma holds.

By (W), the assumption for this case that $w^3 \geq 5/6$, and Lemma 24, the weight of every task in $\hat{\tau}^3$ is greater than $1/2$. Let U_i be a subtask in τ'. Then, because $wt(U) > 1/2$, by Lemma 1, U_i spans at most three slots, *i.e.* $d(U_i) - r(U_i) \leq 3$, which by (60) implies that $t_d - 1 \leq t + 2$. If $t_d - 1 = t + 1$, then (54) establishes the lemma. Therefore, assuming that $t_d - 1 = t + 2$, we first derive an expression for the total lag at $t + 2$.

Let V_k be a subtask in τ', V_j its predecessor, and V_l, a later subtask. Then, by (60) and (3),

$$d(V_j) \leq t + 1 \land d(V_l) > t_d - 1. \tag{61}$$

Because no subtask of τ' is in $\hat{\tau}^3$, $V_k \notin \hat{\tau}^3$ and by (61) and (B2), no subtasks that are later than V_k are in $\hat{\tau}^3$. Thus, there does not exist any subtask of V, whose window overlaps slot $t + 1$, and hence, V is inactive in slot $t + 1$. Therefore, the share of V in $t + 1$ is zero in the ideal system, and hence, the total share of the remaining tasks of $\hat{\tau}^3$ in $t + 1$ is given by

$$\begin{aligned}
&\sum_{T \in \hat{\tau}^3 - \{V\}} share(T, t+1) \\
&\leq \sum_{T \in \hat{\tau}^3 - \{V\}} wt(T) && \text{(by (10))} \\
&= \sum_{T \in \hat{\tau}^3} wt(T) - wt(V) \\
&= \hat{M}^3 - wt(V) && \text{(by (16))}
\end{aligned}$$

$$\leq \quad \hat{M}^3 - 2/3. \quad \text{(by (W), } w^3 \geq 5/6\text{, and Lemma 26)} \tag{62}$$

By (12), the total lag of $\hat{\tau}^3$ at $t+2$ is given by

$$LAG(\hat{\tau}^3, t+2) \leq LAG(\hat{\tau}^3, t+1) + \sum_{T \in \hat{\tau}^3 - \{V\}} share(T, t+1) - \sum_{T \in \hat{\tau}^3} S(T, t+1). \tag{63}$$

Because $t+2 = t_d - 1$, we have $t+1 = t_d - 2$, and hence, by Lemma 56, there are no holes in $t+1$. Therefore, the second summation in the last expression above is $\lfloor \hat{M}^3 \rfloor$ or $\lfloor \hat{M}^3 \rfloor + 1$, depending on whether $t+1$ is tight or non-tight. (It is not pseudo-tight by Lemma 56.) If $t+1$ is non-tight, then by (59) and (63), $LAG(\hat{\tau}^3, t+2) < 3 \cdot \lfloor \hat{M}^3 \rfloor + \hat{M}^3 - 2/3 - \lfloor \hat{M}^3 \rfloor - 1$, which by Lemmas 25 and 12 yields $LAG(\hat{\tau}^3, t+2) < 3 \cdot \lfloor \hat{M}^3 \rfloor$, and completes the proof for this subcase. Therefore, in what follows, we assume the following.

$$t+1 \text{ is tight.} \tag{64}$$

Hence, by (59) and (63), we have $LAG(\hat{\tau}^3, t+2) < 3 \cdot \lfloor \hat{M}^3 \rfloor + \hat{M}^3 - 2/3 - \lfloor \hat{M}^3 \rfloor$, which by Lemma 25 implies that

$$LAG(\hat{\tau}^3, t+2) < 3 \cdot \lfloor \hat{M}^3 \rfloor + w^3 - 2/3. \tag{65}$$

We consider two subcases depending on w^3.

Subcase 1(b)-(i): $t_d - 1 = t+2 \ \wedge \ w^3 \leq 2/3$. For this subcase, $LAG(\hat{\tau}^3, t+2) < 3 \cdot \lfloor \hat{M}^3 \rfloor$, from (65) and the assumption that $w^3 \leq 2/3$.

Subcase 1(b)-(ii): $t_d - 1 = t+2 \ \wedge \ w^3 > 2/3$. For this subcase, we first observe that by (65) and because $w^3 < 1$, $LAG(\hat{\tau}^3, t+2) < 3 \cdot \lfloor \hat{M}^3 \rfloor + 1$. We next show that one of the slots in $[t+2, t+4]$ is non-tight. If either of $t+2$ and $t+3$ is non-tight, the proof for the subcase is complete. Therefore, assume that they are both tight. By Lemma 49, $lag(D^3, t+1) > -1$. Therefore, by (11), (64), and our assumtion that $t+2$ and $t+3$ are tight, $lag(D^3, t+4) > -1 + 3 \cdot w^3$, which by the assumption of this subcase that $w^3 > 2/3$ implies that $lag(D^3, t+4) > 1$. Therefore, by Lemma 50, $t+4$ is non-tight. Because $t_d - 1 = t+2$, we have $t+4 = t_d + 1$, and thus, $t_d + 1$ is non-tight.

Case 2: There are no holes in t. This case can be proved in a manner that is exactly similar to the proof for Case 2 of Lemma 45. □

By $LAG(\hat{\tau}^3, 0) = 0$ and $lag(D^3, 0) = 0$, Lemmas 55–58 above can be seen to contradict Lemma 48. Therefore, (B1) is false, or the tardiness of D^m under I-EPDF is at most two.

Thus, we have established that I-EPDF ensures a tardiness of at most three for tasks in τ^3 and a tardiness of at most two for the donor tasks with weights at least $5/6$, and added to Class 3. As mentioned earlier, the same proof can be generalized to apply to Class c, where $c > 3$.

C.3.2 Case B: $1/2 \leq w^m < 5/6$

If $1/2 \leq w^m < 5/6$, in order to ensure that the tardiness bound of Class m is met if the weight of the donor task added to Class m exceeds $5/6$, a tardiness of at most two needs to be ensured for D^m. However, the assumption that $w^c > 1/2$ and hence, that the tardiness of D^c is at most $c-2$ will not hold. In the case of Class 3, while the tardiness of D^3 is at most one, it cannot be assumed that $2 \cdot w^3$ is at least one, and hence, the proof presented above does not hold for all cases. Hence, in this subsection, we briefly show that I-EPDF ensures a tardiness of c to tasks in τ^c and a tardiness of $c-1$ to D^m in this case also.

We prove this case for Class 3, and for simplicity of notation, only for Class 4, among higher classes. Definitions 6 and 7, (B1), and (B2) of the previous case hold for this case also. However, because the difference between the tardiness of Class c and that of its dummy task D^c may not be greater than one, we establish the inequality

$$LAG(\hat{\tau}^3, t_d - 1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t_d - 1) + w^3, \tag{66}$$

which is slightly stronger than (58). However, there are certain cases where we distinguish Class 3 from other higher-indexed classes, where we establish (58) for Class 3 and (66) for higher-indexed classes (by establishing it for Class 4). (Recall that it is sufficient to establish (58) for Class 3 because the difference in the tardiness bounds of $\hat{\tau}^3$ and D^3 is guaranteed to be two regardless of the weight of the donor tasks added to Class 3.)

If (66) does not hold, then it should be the case that $LAG(\hat{\tau}^3, t) \geq 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t) + w^3$, for some $t \leq t_d - 1$. As illustrated earlier, the inequality can cease to hold only at the end of a pseudo-tight slot or a non-tight slot, t with holes. Because $0 \leq lag(D^3, t+1) < 1$ for such t, we will only show that $LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t) + w^3$, as opposed to giving a full proof.

For the rest of the proof, the following hold.

(D) $1/2 \leq w^m < 5/6$.
(E) $w^3 < 1/2$.

Lemma 59 *Let $t < t_d - 1$ be a psudo-tight slot or a slot with holes, and let $D^m \in A(t)$. Then, $LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + w^3$.*

Proof Sketch: By techniques used earlier, it can be easily shown that the total lag of $\hat{\tau}^3$ at $t+1$ is given by

$$LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 3 + 3 \cdot w^m. \tag{67}$$

We next show that $w^3 > w^m - 1/3$.

Claim 1 $w^3 \geq w^m - 1/3$.

By Lemma 20, the sum of the weights of the donor tasks added to Class 3 and higher is equal to $1 - f^c + w^c$. Because D^m is added to Class 3, we have $w^m \leq 1 - f^3 + w^3$, which implies that $w^3 \geq w^m - 1 + f^3$. By (L3), $f^3 > 2/3$, and hence, $w^3 > w^m - 1/3$. □

Therefore, if the lemma is false, then (67) would imply that $3 \cdot \lfloor \hat{M}^3 \rfloor - 3 + 3 \cdot w^m \geq 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + w^3$, which by Claim 1 would imply that $-3 + 3 \cdot w^m > -1 + w^m - 1/3$, or that $2 \cdot w^m > 2 - 1/3$, which implies that $w^m > 5/6$, which is a contradiction to (D). □

Lemma 60 *Let $t < t_d - 1$ be a pseudo-tight slot or a slot with holes, and let $B(t) \neq \emptyset$. Then, $LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t+1) + w^3$.*

Proof: By (47), $LAG(\hat{\tau}^3, t+1) = \sum_{T \in A(t) \cup B(t) \cup I(t)} lag(T, t+1)$, which by Lemmas 39 and 40 gives $LAG(\hat{\tau}^3, t+1) \leq \sum_{T \in A(t)} lag(T, t+1)$. Because $B(t)$ is not empty, by Lemma 42, there exists a task W in $A(t)$ such that $lag(W, t+1) < wt(W)$. By Lemma **??**, the lag of the remaining tasks is at most three. Because t is either pseudo-tight or contains holes, $|A(t)| \leq \lfloor \hat{M}^3 \rfloor$. Therefore, by (W), $LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 3 + 3/4 = 3 \cdot \lfloor \hat{M}^3 \rfloor - 9/4$, which is less than $3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t) + w^3$.

□

Lemma 61 *Let $t < t_d - 1$ be a pseudo-tight slot or a slot with holes, and let $D^m \notin A(t)$. Let the total number of tasks in τ^3 (tasks in Class 3 excluding the donor tasks) be greater than $\lfloor \hat{M}^3 \rfloor$, and let $LAG(\hat{\tau}^3, t) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t) + w^3$. Then, $LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t+1) + w^3$.*

Proof: If $B(t) \neq \emptyset$, the required result follows from Lemma 60. Therefore, assume $B(t) = \emptyset$ for the rest of the proof. We consider the following two cases.

Case 1: $|A(t)| < \lfloor \hat{M}^3 \rfloor$**.** In this case, by (47), $LAG(\hat{\tau}^3, t+1) = \sum_{T \in A(t) \cup B(t) \cup I(t)} lag(T, t+1)$, which by Lemmas 39 and 40 gives $LAG(\hat{\tau}^3, t+1) \leq \sum_{T \in A(t)} lag(T, t+1)$. By Lemma 53, $lag(T, t+1) < 3$, for $T \in A(t)$, which by the assumption for this case yields $LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 3$, which is less than $3 \cdot \lfloor \hat{M}^3 \rfloor - 1 - lag(D^3, t+1) + w^3$ (because $lag(D^3, t+1) > -1$).

Case 2: $|A(t)| = \lfloor \hat{M}^3 \rfloor$**.** By (12), $LAG(\hat{\tau}^3, t+1) = LAG(\hat{\tau}^3, t) + \sum_{T \in \hat{\tau}^3} (share(T, t) - S(T, t))$. Because the share of tasks in $I(t)$ is zero, and the number of subtasks scheduled at t is $\lfloor \hat{M}^3 \rfloor$, by (47), $LAG(\hat{\tau}^3, t+1) = LAG(\hat{\tau}^3, t) + \sum_{T \in A(t) \cup B(t)} share(T, t) - \lfloor \hat{M}^3 \rfloor$, which by (10) and (47) gives

$$LAG(\hat{\tau}^3, t+1) \leq LAG(\hat{\tau}^3, t) + \sum_{T \in \hat{\tau}^3 - I(t)} wt(T) - \lfloor \hat{M}^3 \rfloor. \quad (68)$$

By the statement of the lemma, $|\tau^3| \geq \lfloor \hat{M}^3 \rfloor + 1$, which by (15) and because $D^m \in \lambda^3$, gives $|\hat{\tau}^3| \geq \lfloor \hat{M}^3 \rfloor + 2$. Because $|A(t)| = \lfloor \hat{M}^3 \rfloor$, and by the assumption for this case, $|B(t)| = 0$, by (47), $|I(t)| \geq 2$, and hence, by (W) and (D), $\sum_{T \in I(t)} wt(T) > 1$, which when substituted in (68) gives $LAG(\hat{\tau}^3, t+1) \leq LAG(\hat{\tau}^3, t) + \hat{M}^3 - 1 - \lfloor \hat{M}^3 \rfloor$, which equals $LAG(\hat{\tau}^3, t) + w^3 - 1$, by Lemma 25. It can be shown that $lag(D^3, t+1) \geq lag(D^3, t) - 1 + w^3$ (as has been done earlier), irrespective of the nature of t, and hence, that $LAG(\hat{\tau}^3, t+1) \leq 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t+1) + w^3$. □

Unless otherwise stated, we assume the following for the lemmas that follow.

(C) $|\tau^c| = \lfloor \hat{M}^c \rfloor$, $c \geq 3$.

Lemma 62 *Let $t < t_d - 1$ be a tight slot with holes. Then $LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 3$.*

Proof: Follows from the fact that $A(t) \leq \lfloor \hat{M}^3 \rfloor - 1$, tardiness of every task in τ^3 is at most three, (W), and Lemma 53. □

Lemma 63 *Let t be a pseudo-tight slot or a non-tight slot with holes. If t is non-tight, then let $d(D_i^3) \leq t$, where D_i^3 is the subtask of D^3 scheduled at t in S^1. Then $lag(D^3, t+1) \geq 0$.*

Proof: Straightforward. □

The two lemmas that follow can be used to infer about the tardiness of the subtask of D^m that was scheduled at the latest time before t, if D^m is inactive at t.

Lemma 64 *Let D_i^m be a subtask of D^m whose release is postponed. Then, the tardiness of the predecessor of D_i^m is zero.*

Proof: Assume not. Let D_h^m be the predecessor of D_i^m. Let t' be the slot in which D_h^m is scheduled. Then, by our assumption, $d(D_h^m) \leq t'$. If there are no IS separations, then by (2), $r(D_{h+1}^m) \leq t'$, and because $w^m > 1/2$, using Lemma 2, it can be easily shown that

$$d(D_{h+1}^m) \leq t' + 2. \quad (69)$$

((69) holds because it can be shown that the length of the window of D_{h+1}^m, $|w(D_{h+1}^m)|$, is two if $b(D_h^m) = 0$ and is at most three if $b(D_h^m) = 1$.) Because D_h^m is scheduled in t', the release time of D_{h+1}^m cannot be postponed at t'. Because

(69) holds, the rules of I-EPDF prohibit its postponement at $t' + 1$ or later. (Refer line 9 in Fig. 5.) Therefore, the assumption that the tardiness of D_h^m is greater than zero is false. $\square$

Lemma 65 *Let t be a slot with holes or a pseudo-tight slot, and let $D^m \in I(t)$. Let D_i^m be the last subtask of D^m scheduled before t. Then $tardiness(D_i^m) = 0$.*

Proof: Because t is a pseudo-tight slot, the release time of the next schedulable subtask is postponed at t. By Lemma 64, the tardiness of its predecessor D_i^m is at most zero. $\square$

Lemma 66 *Let t be a tight slot in schedule S^c for $\hat{\tau}^c$, where $c = 3$ or $c = 4$, and let $D^m \in A(t)$. Let the tardiness of the subtask of D^m scheduled at t be k. If $t + 1$ is non-tight or $D^m \notin A(t + 1)$, then $LAG(\tau^c, t + 2) < c \cdot \lfloor \hat{M}^c \rfloor - c + (k + 2)\frac{c}{c+1}$.*

Proof: By (C), $\tau^c = \lfloor \hat{M}^c \rfloor$. Therefore, because t is tight and D^m is scheduled at t, there is at least one task T in τ^c that is not scheduled at t. This implies that the deadline of the next eligible subtask T_i of T is not less than the deadline of any other subtask that is scheduled at t, specifically, that of subtask D_j^m of D^m scheduled at t. The conditions specified in the lemma imply that, if eligible, T_i can be scheduled at $t+1$. By the previous argument, the tardiness of T at $t+2$ will at most be $k + 2$. The tardiness of all other tasks is at most c resulting in the bound stated in the lemma. $\square$

Lemma 67 *Let t be a pseudo-tight slot in schedule S^c for $\hat{\tau}^c$, where $c = 3$ or $c = 4$. and let $D^m \in A(t)$. If $t+1$ is non-tight or $D^m \notin A(t+1)$, then $LAG(\tau^c, t+2) < c\cdot \lfloor \hat{M}^c \rfloor - c$.*

Proof: The reasoning is similar to that of Lemma 66. Because t is pseudo-tight, $|\tau^c| = \lfloor \hat{M}^c \rfloor$, and D^m is scheduled at t, there is at least one task T that is ineligible at t. It is easy to see that $lag(T, t + 1) \leq 0$. Because D^m is not scheduled at $t + 1$ or $t + 1$ is non-tight, T can be scheduled at $t + 1$, if eligible. Therefore, by (11) $lag(T, t + 2) = lag(T, t + 1) - 1 + wt(T) < 0$. The tardiness of the remaining tasks in τ^c is at most c at $t + 2$, and therefore, their lags are at most c, which gives the bound stated in the lemma. $\square$

Lemma 68 *Let D_i^m be a subtask of D^m scheduled at some time t. Let $tardiness(D_i^m) = k$ and let D_h^m be the predecessor of D_i^m. Then, $tardiness(D_h^m) \leq k + 1$.*

Proof: If $tardiness(D_h^m) = 0$, then the proof is complete. Therefore, assume that $tardiness(D_h^m) > 0$, which by Lemma 64 implies that there are no IS or GIS separations between D_h^m and D_i^m. By the statement of the lemma, subtask D_i^m is scheduled at t with a tardiness of k. Therefore,

$$d(D_i^m) = t + 1 - k. \tag{70}$$

We consider two cases based on the length of the window of D_i^m.

Case 1: $|w(D_i^m)| = 2$**.** By (70), we have $r(D_i^m) = t-k-1$. Because there are no IS separations between D_h^m and D_i^m, by (2), we have $d(D_h^m) = t - k - 1$ or $d(D_h^m) = t - k$. Because $t - 1$ is the latest time that D_h^m is scheduled, if the former holds, then $tardiness(D_h^m) \leq t-(t-k-1) = k+1$, and if the latter holds, then $tardiness(D_h^m) \leq t-(t-k) = k$.

Case 2: $|w(D_i^m)| = 3$**.** In this case, we have $r(D_i^m) = d(D_i^m)-3 = t-k-2$, and by parts (d) and (b) of Lemma 2, that $d(D_h^m) = t - k - 1$. Therefore, because D_h^m can be scheduled no later than $t - 1$, we have $tardiness(D_h^m) \leq k + 1$. $\square$

Lemma 69 *Let $t < t_d - 1$ be a pseudo-tight slot or a non-tight slot with holes, and let $D^m \in I(t)$. Let $t' < t$ be the latest slot in which D^m is scheduled before t, and let t' be tight or pseudo-tight. Then, $LAG(\hat{\tau}^3, t + 1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t + 1) + w^3$.*

Proof: Let τ' denote the set of all tasks that are in τ^3 and scheduled at t'. Then,

$$\tau' = \{T | T \in \tau^3 \wedge T \in A(t')\}. \tag{71}$$

Then, by (47),

$$LAG(\hat{\tau}^3, t + 1) = lag(D^m, t + 1) + \sum_{T \in \tau^3 - \tau'} lag(T, t + 1) + \sum_{T \in \tau'} lag(T, t + 1). \tag{72}$$

We consider two cases depending on whether t' is tight or pseudo-tight.

Case 1: t' is pseudo-tight. Because t' is pseudo-tight, $|A(t')| \leq \lfloor \hat{M}^3 \rfloor$. Therefore, because D^m is scheduled at t', by (C),

$$|\tau'| \leq \lfloor \hat{M}^3 \rfloor - 1. \tag{73}$$

Because D^m is not scheduled after t', by (C), if eligible, every task in τ^3, can be scheduled at each slot. Because every task in $\tau^3 - \tau'$ is either in $I(t')$, or its critical subtask at t' has a tardiness of zero, tardiness is zero for every subtask of a task in $\tau^3 - \tau'$ that is scheduled after t'. Because D^m is inactive at t, $lag(D^m, t + 1) = 0$, which by (72), (W), and Lemmas 39, 40, 52, and 53 yields $LAG(\hat{\tau}^3, t + 1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 9/4$, which is less than $3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t + 1) + w^3$.

Case 2: t' is tight. Let D_i^m be the subtask of D^m that is scheduled at t'. Then, by Lemma 65, $tardiness(D_i^m) = 0$, which implies that $d(D_i^m) \geq t'+1$. By the conditions of the lemma and this subcase, $k = |\tau'| \leq \lfloor \hat{M}^3 \rfloor - 1$. Let W be a task in $\tau^3 - \tau'$, and let W_j be the next eligible subtask of W. Because D_i^m is scheduled at t', and has its deadline at or after $t'+1$, we have $d(W_j) \geq t'+1$. Because D^m is not scheduled after t', by (C), every task of τ^3, if eligible, can be scheduled at every slot following t'. Therefore, tardiness is at most one for every subtask of a task in $\tau^3 - \tau'$ that is scheduled after t'. Because D^m is inactive at t, $lag(D^m, t+1) = 0$, which by (72), (W) and Lemmas 39, 40, 52, and 53 yields $LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 3/2$, which by Lemma 63 is less than $3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + lag(D^3, t+1) + w^3$. $\square$

In the three lemmas that follow, we assume that if t is non-tight, then $d(D_i^c) \leq t+1$, where D_i^c is the subtask of D^c scheduled at t in S^{Sup_c}, where $c = 3$ or $c = 4$, as the case may be. The proof for when $d(D_i^c) > t+1$ is the same as the proof for Subcase 1(b) of Lemma 58.

Lemma 70 *Let $t < t_d - 1$ be a pseudo-tight slot or a non-tight slot with holes, and let $D^m \in I(t)$. Let the slot in which the last subtask of D^m is scheduled before t be non-tight. Then, $LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + 2 \cdot w^3$.*

Proof: We prove this lemma by considering the earliest slot t' before t where the following hold. (i) Subtask D_i^m of D^m is scheduled at t' and $tardiness(D_i^m) = 0$. (ii) t' is non-tight. (iii) Every slot where a subtask D_j^m of D^m that is released later than D_i^m is scheduled is non-tight, and $tardiness(D_j^m) = 0$. That t' exists can be verified from the statement of the lemma.

Let D_h^m be the predecessor of D_i^m. (D_h^m exists. Otherwise, every task in τ^3 is schedulable (if eligible) in every slot in $[0, t]$ and will have a tardiness of zero at $t+1$.) Then, by (i)–(iii), D_h^m is either scheduled in a slot that is not non-tight or $tardiness(D_h^m) > 0$ or both hold. By Lemmas 64 and 68, $tardiness(D_h^m) \leq 1$. Let D_h^m be scheduled at t_1. Then, $t_1 \leq t' - 1$. We consider the following cases based on the nature of t_1.

Case 1: t_1 is tight or pseudo-tight. In this case, by Lemmas 66 and 67, $LAG(\tau^3, t_1 + 1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 3/4$ (substituting $c = 3$ and $k = 1$). Because D^m is only scheduled in a non-tight slot after t_1, by (C), every task in τ^3, if eligible, can be scheduled at every instant after t_1. Therefore, $LAG(\tau^3, t+1)$ and hence, $LAG(\hat{\tau}^3, t+1)$ is less than $3 \cdot \lfloor \hat{M}^3 \rfloor - 3/4$, which is less than $3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + 2 \cdot w^3 < 3 \cdot \lfloor \hat{M}^3 \rfloor - 2/3$. (By Lemma 14, $w^3 \geq 1/6$.)

Case 2: t_1 is non-tight.

We now consider the latest time $\hat{t}$ before t_1, that D^m is scheduled in a tight slot with a tardiness of two. (Again, such a $\hat{t}$ exists. Otherwise, every task in τ^3 is schedulable in every slot in $[0, t]$.) Therefore, $\hat{t} \leq t' - 2$. Because, D^m is only scheduled in a non-tight slot after $\hat{t}$, by (C), every task in τ^3 is scheduled in every slot if eligible. We consider two cases depending on whether all tasks in τ^3 are scheduled in every slot in $[\hat{t}+1, t]$. If some task in τ^3 is not scheduled in a slot, then its tardiness is zero, at t, and hence $LAG(\hat{\tau}^3, t+1) < 3 \cdot \lfloor \hat{M}^3 \rfloor - 2$. On the other hand, if every task is scheduled in every slot then $LAG(\tau^3, t+1) = LAG(\tau^3, \hat{t}+2) + \sum_{u=\hat{t}+2}^{t} \sum_{T \in \tau^3} (share(T, u) - S(T, u))$. Because every task in τ^3 is scheduled in every u, $\sum_{T \in \tau^3} S(T, u) = \lfloor \hat{M}^3 \rfloor$. By (10) and (13), $\sum_{T \in \tau^3} share(T, u) \leq M^3$, for every u. Therefore, the summation in the expression for LAG just above is $M^3 - \lfloor \hat{M}^3 \rfloor$. From $\hat{M}^3 = M^3 + w^m$ and $\hat{M}^3 - \lfloor \hat{M}^3 \rfloor = w^3$, it easily follows that $M^3 - \lfloor \hat{M}^3 \rfloor = w^3 - w^m$. Because $t > \hat{t} + 2$, we have $t - \hat{t} - 1 \geq 2$, and hence $LAG(\tau^3, t+1) < LAG(\tau^3, \hat{t}+2) + 2 \cdot (w^3 - w^m)$. It can easily be shown that $LAG(\tau^3, \hat{t}+2) < 3 \cdot \lfloor \hat{M}^3 \rfloor$. If $3 \cdot \lfloor \hat{M}^3 \rfloor - 2 \cdot (w^m - w^3) > 3 \cdot \lfloor \hat{M}^3 \rfloor - 1 + 2 \cdot w^3$, then it implies that $-2 \cdot (w^m - w^3) > -1 + 2 \cdot w^3$, or $2 \cdot (w^m - w^3) < 1 - 2 \cdot w^3$, which implies that $w^m < 1/2$, which is a contradiction to (D). $\square$

Finally, we prove the above lemma for Class 4. (A separate proof for Class 4 is necessary because we need to show that $LAG(\hat{\tau}^4, t+1) < 4 \cdot \lfloor \hat{M}^4 \rfloor - 1 + w^4$.) Based on earlier discussion, we have the following assumptions.

(D) $1/2 \leq w^m < 5/6$. **(E)** $w^4 < 1/2$.

We begin by proving a slightly stronger tardiness bound for $\hat{\tau}^4$, when the number of tasks in τ^4 is one, *i.e.*, $\lfloor \hat{M}^4 \rfloor = 1$. In other words, we show that when $\lfloor \hat{M}^4 \rfloor = 1$, I-EPDF ensures a tardiness of at most three for $\hat{\tau}^4$. By arguments similar to what have been used several times until now, to show that I-EPDF ensures a tardiness of at most three to $\hat{\tau}^4$ (*i.e.*, to the lone task in τ^4 and D^m), it suffices to show that $LAG(\hat{\tau}^4, \hat{t}) \leq 3 \cdot \lfloor \hat{M}^4 \rfloor + lag(D^4, \hat{t})$, *i.e.*, $LAG(\hat{\tau}^4, \hat{t}) \leq 3 + lag(D^4, \hat{t})$, for all $\hat{t}$. For brevity, we only show that this lag bound is maintained across pseudo-tight slots and non-tight slots with holes.

Lemma 71 *Let $t < t_d - 1$ be a pseudo-tight slot or a non-tight slot with holes. If $\lfloor \hat{M}^4 \rfloor = 1$ and $LAG(\hat{\tau}^4, t) \leq 3 \cdot \lfloor \hat{M}^4 \rfloor + lag(D^4, t)$, then $LAG(\hat{\tau}^4, t+1) \leq 3 \cdot \lfloor \hat{M}^4 \rfloor$.*

Proof: Let T be the only task in τ^4. Then, by (D) and Lemma 24, there are only two tasks in $\hat{\tau}^4$, T and D^m, and by Lemma 25 and (16), the following holds.

$$
\begin{aligned}
1 + w^4 &= wt(T) + w^m \\
\Rightarrow w^m &= 1 + w^4 - wt(T) \\
\Rightarrow w^m &< 1 + 1/2 - 3/4 && \text{(from (E) and (W))} \\
\Rightarrow w^m &\leq 3/4. && (74)
\end{aligned}
$$

Because t is a pseudo-tight slot or a non-tight slot with holes, at most one task is scheduled in t. We consider two cases depending on whether D^m is scheduled in t.

Case 1: D^m is scheduled in t. For this case, by Lemma 53 and (74), $lag(D^m, t+1) < 4 \cdot w^m = 3$, and by Lemmas 39 and 40, $lag(T, t+1) \leq 0$. Therefore, $LAG(\hat{\tau}^4, t+1) < 3$, which proves the lemma.

Case 2: D^m is not scheduled in t. For this case, we consider two subcases depending on the nature of the slot in which the last subtask of D^m is scheduled before t.

Subcase 2(a): D^m was last scheduled in a tight or pseudo-tight slot before t. As it was shown in Lemma 69, it can be shown that the tardiness of the subtask of T scheduled at t is at most one, and hence, that $LAG(\hat{\tau}^4, t+1) < 8/5$.

Subcase 2(b): D^m was last scheduled in a non-tight slot before t. For this case, we consider the earliest slot t' before t in which the following hold. **(i)** Subtask D^m_i of D^m is scheduled at t' and $tardiness(D^m_i) = 0$. **(ii)** t' is non-tight. **(iii)** Every slot where a subtask D^m_j of D^m that is released later than D^m_i is scheduled is non-tight and $tardiness(D^m_j) = 0$. From the statement of the lemma it is easy to see that such a t' exists.

Let D^m_h be the predecessor of D^m_i. Then, by (i)–(iii), D^m_h is either scheduled in a slot that is not non-tight or $tardiness(D^m_h) > 0$ or both hold. By Lemmas 64 and 68, $tardiness(D^m_h) \leq 1$. Let D^m_h be scheduled at t_1. Then, $t_1 \leq t' - 1$. We consider the following subcases based on the nature of t_1.

Subcase 2(b)-i: t_1 is tight or pseudo-tight. In this case, by Lemma 53, $lag(T, t_1+1) < 3 \cdot (4/5) = 12/5$. By (i)–(iii) and $\lfloor \hat{M}^4 \rfloor = 1$, T can be scheduled in every slot after t_1. Therefore, it can be shown that $lag(T, t+1)$, and hence, $LAG(\hat{\tau}^4, t+1) < 3$.

Subcase 2(b)-ii: t_1 is non-tight. For this case, we consider the predecessor D^m_g of D^m_h. By Lemmas 64 and 68, $tardiness(D^m_g) \leq 2$. If $tardiness(D^m_g) \leq 1$, then the lemma can be shown to hold using the arguments used for Case 1. Therefore, for the rest of this case, asume that $tardiness(D^m_g) = 2$. Because D^m_h is scheduled at t_1, D^m_g cannot be scheduled later than $t_1 - 1$, *i.e.*, $t' - 2$. Therefore, the tardiness of the subtask of T scheduled at t_1 can be as high as three, and hence, by Lemma 53 and (W), $lag(T, t_1+1) < 4 \cdot (4/5)$. Again, by (i)–(iii) and $\lfloor \hat{M}^4 \rfloor = 1$, T can be scheduled at every slot later than t_1. Therefore, because there are at least two slots in the interval $[t_1+1, t]$, $lag(T, t+1) \leq lag(T, t_1) - 2(1 - wt(T)) < 16/5 - 2 \cdot (1/5) = 14/5 < 3$. □

We next prove the counterpart of Lemma 70 for Class 4 when $\lfloor \hat{M}^4 \rfloor > 1$.

Lemma 72 *Let $t < t_d - 1$ be a pseudo-tight slot or a slot with holes, and let $D^m \in I(t)$. Let $\lfloor \hat{M}^4 \rfloor > 1$ and the slot in which the last subtask of D^m is scheduled before t be non-tight. If $LAG(\hat{\tau}^4, t) < 4 \cdot \lfloor \hat{M}^4 \rfloor - 1 + lag(D^4, t) + w^4$, then $LAG(\hat{\tau}^4, t+1) < 4 \cdot \lfloor \hat{M}^4 \rfloor - 1 + w^4$.*

Proof:

As with Lemma 70, we prove this lemma by considering the earliest slot t' before t in which the following hold. **(P1)** Subtask D^m_i of D^m is scheduled at t' and $tardiness(D^m_i) = 0$. **(P2)** t' is non-tight. **(P3)** Every slot where a subtask D^m_j of D^m that is released later than D^m_i is scheduled is non-tight and $tardiness(D^m_j) = 0$. By our assumptions, we have

$$t' \leq t - 1. \tag{75}$$

We first establish the following claim.

Claim 2 *Let D^4_i be the subtask of D^4 scheduled at t' in $\hat{\tau}^3$. Then, $d(D^4_i) \leq t' + 1$.*

Proof: Assume to the contrary that $d(D^4_i) > t'+1$. Then, it is easy to see that $lag(D^4_i, t'+1) < 0$. Therefore, by (75) and the statement of the lemma, $LAG(\hat{\tau}^4, t'+1) < 4 \cdot \lfloor \hat{M}^4 \rfloor - 1 + w^4$. By (C), (P2) and (P3), every task in τ^4, if eligible, can be scheduled at t', and in every slot following t' until t, *i.e.*, in every slot in the interval $[t', t]$. If some task in τ^4 is not scheduled in some slot in $[t', t]$, then it should be the case that the task is either inactive or is ineligible at that slot, using which it can be concluded that its tardiness at $t+1$ is zero. Hence, it would easily follow that $LAG(\hat{\tau}^4, t+1) < 4 \cdot \lfloor \hat{M}^4 \rfloor - 3$. On the other hand, if every task in τ^4 is scheduled in every slot in $[t', t]$, then the lag of each task at $t+1$ is less than its lag at $t'+1$, *i.e.*, $LAG(\hat{\tau}^4, t+1) < 4 \cdot \lfloor \hat{M}^4 \rfloor - 1 + w^4$. □

Let D^m_h be the predecessor of D^m_i. Then, by (P1)–(P3), D^m_h is either scheduled in a slot that is not non-tight or $tardiness(D^m_h) > 0$ or both hold. By Lemmas 64 and 68, $tardiness(D^m_h) \leq 1$. Let D^m_h be scheduled at t_1. Then, $t_1 \leq t' - 1$. We consider the following cases based on the nature of t_1.

Case 1: t_1 is tight or pseudo-tight. In this case, by Lemmas 66 and 67, and because $tardiness(D^m_h) \leq 1$, we have $LAG(\tau^4, t_1+1) < 4 \cdot \lfloor \hat{M}^4 \rfloor - 8/5$. By (C), every task in τ^4 can be scheduled in every slot following t_1 . Therefore, it can be shown that $LAG(\hat{\tau}^4, t+1) < 4 \cdot \lfloor \hat{M}^4 \rfloor - 8/5$.

Case 2: t_1 is non-tight. For this case, we consider the predecessor D^m_g of D^m_h. By Lemmas 64 and 68, $tardiness(D^m_g) \leq 2$. If $tardiness(D^m_g) \leq 1$, then the lemma can be shown to hold using the arguments used for Case 1. Therefore, for the rest of this case, assume that $tardiness(D^m_g) = 2$. Because D^m_h is scheduled at t_1, D^m_g cannot be scheduled later than $t_1 - 1$, *i.e.*, $t' - 2$. We consider two subcases based on the nature of t_2, the slot in which D^m_g

is scheduled. By the discussion above, we have

$$t_2 \leq t' - 2. \quad (76)$$

Subcase 2(a): t_2 **is tight or pseudo-tight.** By Lemmas 64 and 68, $LAG(\tau^4, t_2 + 2) < 4 \cdot \lfloor \hat{M}^4 \rfloor - 4/5$ (because it can be shown that the tardiness of the subtask of task T not scheduled at t_2 is at most three). Again, by (C), every task in τ^4 can be scheduled at every slot that follows t_2. By (75) and (76), we have at least two slots in the interval $[t_2 + 2, t]$. Therefore, $LAG(\tau^4, t + 1) \leq LAG(\tau^4, t_2 + 2) - 2(\lfloor \hat{M}^4 \rfloor - M^4)$. Because $\lfloor \hat{M}^4 \rfloor - M^4$ can be shown to be equal to $w^m - w^4$, it follows that $LAG(\tau^4, t + 1) \leq LAG(\tau^4, t_2 + 2) - 2(w^m - w^4)$. If $LAG(\tau^4, t + 1)$ is greater than or equal to $4 \cdot \lfloor \hat{M}^4 \rfloor - 1 + w^4$, then it implies that $-4/5 - 2(w^m - w^4) > -1 + w^4$, or $w^m < 1/5 + w^4/2$, which by (E) implies that $w^m < 9/20$, which contradicts (D).

Subcase 2(b): t_2 **is non-tight.** For this subcase, we consider the latest time t_3 before t_2, that D^m is scheduled in a tight slot with a tardiness of three. Because $t_2 \leq t' - 2$, we have

$$t_3 \leq t' - 3. \quad (77)$$

We consider two more subcases based on t_3.

Subcase 2(b)-i: $t_3 < t' - 3$. By (B2), the tardiness of every subtask that is scheduled at $t_3 + 1$ is at most four. Because $t_3 < t' - 3$ and $t' < t$, there are at least five slots in the interval $[t_3 + 1, t + 1]$. Because every task in τ^4 can be scheduled in every slot that follows t_3, and by (W), the weight of every task in τ^4 is at most $4/5$, it can be shown that the tardiness of the subtask of every task scheduled at t is at most three. By Lemma 53, this would imply that $LAG(\tau^4, t + 1) < 16 \cdot \lfloor \hat{M}^4 \rfloor / 5$. If $LAG(\tau^4, t + 1) > 4 \cdot \lfloor \hat{M}^4 \rfloor - 1 + w^4$, then it would imply that $16 \cdot \lfloor \hat{M}^4 \rfloor / 5 > 4 \cdot \lfloor \hat{M}^4 \rfloor - 1 + w^4$, *i.e.*, $w^4 < 1 - 4 \cdot \lfloor \hat{M}^4 \rfloor / 5$. Because $\lfloor \hat{M}^4 \rfloor > 1$, this would imply that $w^4 < -3/5$, which is false.

Subcase 2(b)-ii: $t_3 = t' - 3$. If $t_3 = t' - 3$, then it would imply that $t_2 = t' - 2$ and $t_1 = t' - 1$. Therefore, by the conditions of this subcase, we have t', $t' - 1$, and $t' - 2$ to be non-tight. We first establish the following claim.

Claim 3 $w^4 \geq 1/3$.

Proof: Assume to the contrary that $w^4 < 1/3$. Let D_i^4 be the subtask of D^4 that is scheduled at t'. Then, by Claim 2, $d(D_i^4) \leq t' + 1$. We prove the claim for $d(D_i^4) = t' + 1$. The proof for $d(D_i^4) < t' + 1$ is similar. If $r(D_i^4) \geq t' - 2$, then $w(D_i^4) \leq 3$, which by Lemma 3 contradicts the assumption that $w^4 < 1/3$. Therefore, assume $r(D_i^4) \leq t' - 3$. Let D_h^4 be the predecessor of D_i^4. Then, by (3) and (4), $d(D_h^4) \leq t' - 2$. If $r(D_h^4) \geq t' - 5$, then $w(D_h^4) \leq 3$, which would contradict our assumption and satisfy the claim. On the other hand, if $r(D_h^4) \leq t' - 6$, then by (3) and (4), $d(D_g^4) \leq t' - 5$, where D_g^4 is the predecessor of D_h^4. Because $D^4 \in \hat{\tau}^3$, $tardiness(D^4) \leq 3$, which implies that the latest time that D_g^4 may be scheduled is $t' - 3$. This would imply that D^4 is not scheduled in at least on slot in $[t' - 2, t']$, *i.e.*, at least one of $[t' - 2, t']$ is not non-tight, which contradicts the conditions for this subcase. Therefore, the assumption that $w^4 < 1/3$ is false. □

By (C), there is at most one task T in τ^4 that is not scheduled in t_3. Every task in $\tau^4 - T$, if eligible, can be scheduled in every slot in $[t_3, t]$. We next show that $tardiness(U, t+1) \leq 3$ for at least one task U in $\tau^4 - T$. If $tardiness(U, t_3 + 1) = 3$, then because U can be scheduled in every slot in $[t_3, t]$, if eligible, its tardiness at $t + 1$ is at most its tardiness at $t_3 + 1$, *i.e.*, $tardiness(U, t + 1) \leq 3$. On the other hand, if $tardiness(U, t_3 + 1) = 4$, then we argue as follows. Because $t_3 = t' - 3 \leq t - 4$, the number of slots spanning the interval $[t_3, t + 1]$ is at least five. Because $wt(U) \leq 4/5$, it can be easily shown that at most four subtasks of U have their deadlines in consecutive slots. This in turn implies that if the subtask U_i of U scheduled at t_3 has a tardiness of four, and U is scheduled in every slot following t_3 up to t, then at most three subtasks that follow U_i can have a tardiness of four. In other words, $tardiness(U, t+1) \leq 3$. Therefore, $LAG(\hat{\tau}^4, t + 1) < 4 \cdot \lfloor \hat{M}^4 \rfloor - 1 + 1/5$, which is less than $4 \cdot \lfloor \hat{M}^4 \rfloor - 1 + 1/3$.

□