Accepted for publication in Real-Time Systems

Dual-Priority versus Background Scheduling : a Path-wise Comparison

BRUNO GAUJAL, NICOLAS NAVET, JORN MIGGE {gaujal,nnavet,jmigge}@loria.fr

LORIA - INPL

ENSEM - 2, Avenue de la forêt de Haye

54516 Vandoeuvre-lès-Nancy - France

Abstract. In this paper two well-known scheduling policies for Real Time Systems, namely Background Scheduling and Dual-Priority are compared in terms of response times for Soft Real Time traffic (SRT). It is proved in the preemptive as well as in the non-preemptive case that, when the SRT traffic is FIFO, the Dual-Priority policy always outperforms Background Scheduling for all instances of SRT tasks. When the SRT traffic is not FIFO but if all tasks are of equal size then, in the non-preemptive case, the average response times is shown to be always better under DP than under BS. As a complementary result, some non-FIFO examples where Background Scheduling behaves better than Dual-Priority for some SRT tasks but also on the average of the SRT response times, are given. The proofs are based on a trajectorial method that may be used for comparing other scheduling policies.

Keywords: Scheduling Algorithm, Dual-Priority, Background Scheduling, Soft Real-Time, Local Area Network.

1. Introduction

Context of the paper In real-time systems, hard and soft timing constraints generally coexist. The problem of jointly scheduling Hard Real-Time (HRT) traffic and Soft Real-Time (SRT) traffic is an important issue in real-time computing and it arises both for tasks (scheduling on the CPU) and messages (scheduling on the network). The problem that we consider is to study some schedules that reduce as much as possible the average response time of SRT traffic while ensuring the timing requirements of HRT traffic to be met. In this study, it is assumed that the HRT traffic is periodic or sporadic, while no assumptions are placed on SRT traffic for most of the paper (a deterministic stability condition under the form of a (σ, ρ) bound [8], is used for implementation issues).

Existing work The simplest strategy for scheduling both SRT traffic and HRT traffic is to schedule the SRT traffic in the "background" (that will be called the background scheduling policy or BS for short), *i.e.* with a lower priority than any HRT tasks or messages. Experiments have shown that it leads to poor performances in terms of responsiveness of SRT traffic (see [3, 11] for the scheduling of tasks and for message scheduling [23, 13]). Several approaches performing better have been developed for the scheduling of tasks.

Existing scheduling schemes include Earliest Deadline Last (EDL, [7]), Deferrable Server (DS, [19]), Priority Exchange (PE, [19]), Extended Priority Exchange (EPE, [29]), Static Slack Stealing algorithm (STS, [18]) and Dynamic Slack Stealing (DSS, [10]). As pointed out in [3] and [9], they all have disadvantages : DS, PE and EPE do not make use of all the available slack time, EDL and DSS are computationally expensive while SSS may require, depending on the task set, to store huge amount of information.
In [9, 11] an elegant and simple alternative, termed the Dual-Priority (DP) policy, has been proposed. In essence, this policy facilitates the responsive execution of SRT tasks by executing all HRT tasks immediately, when there are no SRT tasks ready, or as late as possible where SRT tasks are ready to be ran.
The DP scheme is applicable over a wide range of problems with non-significant overheads and no restrictive hypotheses except for the knowledge of the worst-case response time for HRT tasks. Experiments published in [9, 11] and later in [3, 4] have shown through a series of simulations that the DP policy is highly effective in terms of responsiveness of SRT tasks in the context of the preemptive scheduling of tasks. From a practical point of view, the implementation of DP is quite straightforward and can be either done at the kernel level such as in the *Monstre* real-time OS [26] (quoted in [3]) or at the application level for instance using ADA constructs as detailed in [6] and [3].
The dual-priority scheme is also usable for message scheduling on a network with bounded access time to the medium and this, even when nodes are not synchronised because the knowledge of the first release time of a periodic source is not mandatory. Its utilisation has been proposed in [30] and it has shown to be very efficient [23, 13] in the context of a CAN (Controller Area Network [15]) based in-vehicle multiplexing system. Recently, the dual-priority strategy has been enhanced in various directions. For the scheduling of tasks, a method for computing tighter bounds on the response time for purely periodic task sets has been published in [3, 4]. In the context of message scheduling, a mechanism that provides probabilistic guarantees to prevent hard real-time frames from missing their deadlines when transmission errors can occur is proposed in [23].

Goal of the paper As previously mentioned, numerous experiments [9, 11, 3, 4, 23, 13] have shown the DP strategy to be very efficient both for the scheduling of tasks and the scheduling of messages. However, the problem of proving, in a precise manner, the efficiency of DP has not been addressed yet. A first important question is whether DP always behaves better than the classical strategy, termed Background Scheduling (BS for short), with which SRT tasks are assigned the lowest priority levels under Fixed Priority Scheduling.
In this paper, we will prove that the response time of each instance of all SRT tasks is always better under the DP policy if and only if the whole SRT traffic is FIFO. This result holds for the preemptive and the non-preemptive case. We will also prove that the average response time of all SRT instances in the non-preemptive case is always smaller under DP if all tasks are of equal size. The comparison between DP and BS will be done in a path-wise manner where a path (also called

a trajectory) of the system is determined by the tasks (resp. messages) activation dates and by their execution (resp. transmission) times. These path-wise comparisons have several advantages. Indeed, they will hold for arbitrary distributions of the arrival process of the SRT traffic and for any increasing functional of the response time, such as the expectation and moments of any order or even logarithm, exponential and Laplace transforms of the response times.

This result is two-fold. On the one hand, it reinforces the results hinted by previous experimental studies showing substantial gain of DP over BS by providing a theoretical basis. On the other hand, it suggests that when the FIFO ordering of SRT tasks is not satisfied, one may experience a surprising phenomenon where some SRT tasks have a lower response time under BS. Indeed, in some cases where the FIFO condition does not hold, we exhibit examples where the response times of some SRT tasks (messages) are better under BS than under DP. Also, several simulations were run under a non-FIFO case for SRT traffic in the context of the CAN network. They also have the same kind of behaviour where the response times of some SRT messages is smaller under BS than under DP.

Organisation of the paper In Section 2, we introduce the framework and the formulation of the problem as well as the notations used in the following. Section 3 deals with the non-preemptive case while Section 4 treats the preemptive case. Section 5 reports some simulations of a CAN priority bus that illustrate the different behaviours coming up in the theoretical parts. Finally, in Section 6, we investigate how to ensure the FIFO ordering of SRT traffic.

2. Framework

The context of this study is the preemptive and the non-preemptive scheduling of mixed SRT and HRT traffic on a shared resource that can be either a processor or a network. Up to Section 4, we will focus on the non-preemptive case. In order to keep a unified vocabulary, we will talk about tasks even though everything, in the non-preemptive case, is transposable to messages.

Throughout this paper, we will use some concepts and ideas introduced in [17] and refined later in [21, 20]. The system under study can be modeled by a finite set of m *recurrent tasks* and one resource that executes the successive *instances* of these tasks. Regarding the timing constraints, the set of tasks can be split into two subsets :

- the HRT tasks, $\mathcal{H} = \{\tau_1, \cdots, \tau_p\}$,
- and the set of SRT tasks $\mathcal{S} = \{\tau_{p+1}, \cdots, \tau_m\}$.

For any task τ_k, $\tau_{k,n}$ denotes the n-th instance of task τ_k, $A_{k,n}$ is the release time of instance $\tau_{k,n}$, and $C_{k,n}$ is the load brought by instance $\tau_{k,n}$. The HRT tasks are assumed to be periodic (resp. sporadic) with a period (resp. minimal inter-arrival time) denoted by T_k for task τ_k.

As for the SRT tasks, no assumption are placed in most of the paper, and SRT traffic can be completely arbitrary. Only in Section 6, a deterministic stability condition will be introduced in order to compute upper bounds on busy periods.
The resource (which capacity is fixed to 1) is shared by all the tasks according to a scheduling policy (S) which assigns the resource to the instances. Under S, $B^S_{k,n}$ is the time when execution of $\tau_{k,n}$ begins and $E^S_{k,n}$ is the time when execution of $\tau_{k,n}$ ends. $\mathbf{R}^S_{k,n}$ is the response time of $\tau_{k,n}$ and by definition $\mathbf{R}^S_{k,n} = E^S_{k,n} - A_{k,n}$. If an instance $\tau_{k,n}$ has been released before time t but has not been completed at time t (*i.e.* $A_{k,n} \leq t < E^S_{k,n}$), then $\tau_{k,n}$ is said to be *pending* at time t.

Each instance of an HRT task τ_k has a *relative deadline* D_k (the absolute deadline for each instance $\tau_{k,n}$ is $A_{k,n} + D_k$). The system is said *feasible* under S if each instance of all HRT tasks meets its deadline. Formally,

$$\forall k, \quad \forall n, \quad \mathbf{R}^S_{k,n} \leq D_k. \tag{1}$$

The resource, scheduled under the policy S, is *busy* at time t if a task is being executed at time t. The busy indicator is the right-continuous function $\beta^S(t)$ equal to 1 whenever the resource is busy at time t and 0 otherwise.
For the sake of simplicity, in the rest of the paper, the following restrictions are placed :

1. Tasks have no jitter in their release date.

2. Context switch latencies are neglected.

3. HRT tasks have deadlines that are less than or equal to their periods.

Jitter in task availability dates (assumption 1) and context switch latencies (assumption 2) can be taken into account in the schedulability analysis as in [5]. The third assumption (deadlines must not be greater than periods) can be relaxed as in [11].

2.1. The Background Scheduling policy (BS)

Under fixed priority scheduling, the priority assignment with which all SRT tasks are given lower priorities than HRT tasks is called the Background Scheduling BS policy. Although experiments [3, 11, 23] have shown that BS performances are poor in terms of responsiveness of SRT tasks under heavy load, BS has two key advantages; it is straightforward to implement and the feasibility of HRT tasks is easily ensured.
Priorities BS is a Fixed Priority scheduling scheme, thus all instances $\tau_{k,n}$ are given a fixed priority level, denoted $\pi^{BS}(k,n)$. We will assume with no loss of generality that all instances are ordered according to their numbering. In other words, $\pi^{BS}(k,n) = (k,n)$, ordered using the alpha-numerical ordering (i.e. $(k,n) < (k',n')$ if $k < k'$ or $k = k'$ and $n < n'$). The priority numbering scheme adopted is "the smaller the number, the higher the priority" and the priority rule says that if $\pi^{BS}(k,n) < \pi^{BS}(k',n')$, then instance $\tau_{k,n}$ has a *higher priority* than instance

$\tau_{k',n'}$ (and $\tau_{k',n'}$ has a lower priority than $\tau_{k,n}$). If several instances of the same task are pending at the same instant, then, the earliest release has priority over all other instances of the same task (the system is FIFO within one task). As previously mentioned, under the BS scheme, all HRT tasks have higher priority than all SRT tasks.

Allocation rule

In the non-preemptive case, the BS policy behaves according to the following rule:

as soon as the resource is not busy, the pending instance
with the highest priority starts being executed.

Note that under the non-preemptive allocation rule, the instance being executed can change only at completion times.

Feasibility is equivalent to $\mathbf{R}_k^{BS} \le D_k \quad \forall \tau_k \in \mathcal{H}$, where $\mathbf{R}_k^{BS}$ is the worst-case response time of task τ_k.

The value of $\mathbf{R}_k^{BS}$ in the non-preemptive case, is the maximum time needed by the task to gain the resource (denoted by I_k) plus C_k. When $D_j < T_j$ for all HRT tasks (the case $D_j > T_j$ is slightly more complicated, see [1]), then τ_k can be delayed by higher priority tasks and by one lower priority task that has already obtained the resource (in the worst-case, it is the execution time of the biggest task with priority lower than τ_k). From [14], we have :

$$\mathbf{R}_k^{BS} = C_k + I_k \tag{2}$$

where I_k is the longest time all higher priority tasks can occupy the bus plus the execution time of the biggest lower priority task. I_k is defined as the limit, when n goes to infinity, of the sequence

$$I_k^0 = 0, \quad I_k^n = \max_{i>k}(C_i) + \sum_{j<k} \left(\left\lfloor \frac{I_k^{n-1}}{T_j} \right\rfloor + 1 \right) C_j. \tag{3}$$

The quantity I_k is computed starting with $I_k^0 = 0$, until convergence or until $I_k^n > D_k - C_k$. In the latter case, τ_k is not guaranteed to respect its deadline and the set of tasks is non-feasible.

2.2. The Dual-Priority policy (DP)

The main difference with BS is that the priority level of the HRT tasks may change dynamically over time. This policy has been proposed in [9, 11]. Under DP, the priority range must be partitioned into three bands: "low hard", "soft", "high hard" in increasing level of priority. All the priorities in the "low hard" range are lower than all "soft" which are lower than "high hard". An HRT task is first queued with a priority within the "low hard" band and later, when it becomes urgent, it will be promoted to the "high hard" range. Instead of executing HRT tasks as soon as they are available, they can be deferred in favour of SRT tasks until they become urgent.

We define the *critical interval* for a HRT instance $\tau_{k,n}$ as the time interval $]A_{k,n}+D_k-\mathbf{R}_k^{BS},\ A_{k,n}+$

$D_k]$. The priorities of HRT instances change over time. For all $k \leq p$ and for all $n \in \mathbb{N}$,

$$\pi^{DP}(k,n,t) = \begin{cases} (k,n) & \text{if } t \text{ is in the critical interval for instance } \tau_{k,n} \\ & \text{("high hard" priority),} \\ (m+k,n) & \text{otherwise ("low hard" priority).} \end{cases} \quad (4)$$

The priorities of SRT tasks τ_k remain fixed over time and stay in the "soft" priority range : $\forall k > p, \forall t, \pi^{DP}(k,n,t) = (k,n)$.
Once the priorities are defined, the policy follows the same rule as BS: *as soon as an execution is completed, the pending instance with the current highest priority starts being executed.*

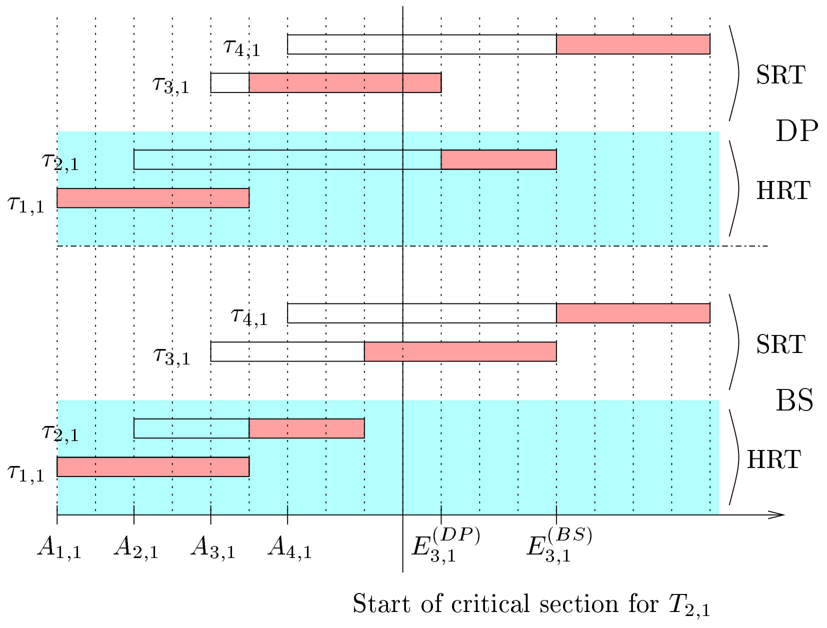

Figure 1. A trajectory under DP and BS policies.

An illustration of a trajectory under BS and DP policies is represented in Figure 1. The rectangles represent the instances of the tasks, with a grey area when the instance is being executed and a white area when it is pending and waiting to be executed. In this example, four tasks are competing for the resource : τ_1 and τ_2 are HRT tasks while τ_3 and τ_4 are SRT tasks (with the notations adopted here, we have $p = 2$ and $m = 4$). The release times are $A_{1,1} = 0, A_{2,1} = 2, A_{3,1} = 4, A_{4,1} = 6$. The execution times are respectively : $C_1 = 5, C_2 = 3, C_3 = 5, C_4 = 4$. At time 0, the only pending instance is $\tau_{1,1}$, therefore, it starts being executed immediately, under DP as well as under BS: $B_{1,1}^{BS} = B_{1,1}^{DP} = 0$. When instance $\tau_{1,1}$ is completed (at time 5), then $\tau_{2,1}$ and $\tau_{3,1}$ are pending. Under BS, their respective priorities are: $\pi^{BS}(2,1) = (2,1)$ and $\pi^{BS}(3,1) = (3,1)$. Therefore, $\tau_{2,1}$ gets the resource: $B_{2,1}^{DP} = 5$. Under DP, their respective priorities are: $\pi^{DP}(2,1,5) = (2+4,1)$ (we assume that instance $\tau_{2,1}$ is not in its critical interval at time 5) and $\pi^{DP}(3,1,5) = (3,1)$. Therefore, $\tau_{3,1}$ gets the resource and $B_{3,1}^{DP} = 5$. Under DP, when $\tau_{3,1}$ is completed at time 10,

$\tau_{2,1}$ and $\tau_{4,1}$ are pending. Their respective priorities are: $\pi^{DP}(2,1,10) = (2,1)$ (it is in its critical interval) and $\pi^{DP}(4,1,10) = (4,1)$. Therefore, $\tau_{2,1}$ gets the resource: $B_{2,1}^{DP} = 10$. Finally, under DP the response time of $\tau_{3,1}$ and $\tau_{4,1}$ is respectively 6 and 11 versus 9 and 11 under BS.

LEMMA 1 *The policy DP is feasible if and only if BS is feasible.*

This lemma is a direct consequence of the computation of a bound on the worst response time of HRT tasks under DP. This computation has been proposed in [11]. Note however that under DP, even if the response time of a HRT task is guaranteed to be smaller than the deadline, it may be increased.

3. Dual-Priority VS Background Scheduling : the non-preemptive case

Numerous simulations suggest that DP improves substantially the average response time of SRT tasks over BS. For the non-preemptive case, simulations were performed in [23, 13] for message scheduling on a CAN (Controller Area Network) priority bus. These simulations use exponential inter-arrival times for SRT messages and measure the average response time and the variance of response times. For both criteria, the gain provided by DP over BS is very important. For instance, the observed gain for average SRT response time ranges from a factor of 2.4 for a 60% total load to a factor of 3.8 for a 95% total load. It is also noteworthy that the dual-priority scheme offers very good resistance to an increase of the network load up to 90%.
However, up to the authors' knowledge, no formal proof of the fact that policy DP actually improves the response times of SRT tasks over BS has been published so far. In this section, we will provide the necessary and sufficient condition under which DP is better than BS on any trajectory. The path-wise method has the advantage that one will be able to compare any increasing functional of the response time for any arrival process of the SRT tasks.

3.1. Busy periods

We define the *total workload* at time t as the left continuous function

$$W(t) = \sum_{(k,n)} C_{k,n} \mathbf{1}_{\{A_{k,n} < t\}} - \int_0^t \beta(u)du, \tag{5}$$

where $\mathbf{1}_B$ is the indicator function over the set B. The function $W(t)$ can be seen as the total amount of work which has arrived before time t and which is still waiting to be done. The first term in Equation (5) is the amount of work arrived before t while the second term is the work done between 0 and t, with time 0 being the availability date of the resource.

LEMMA 2 *The total workload is the same under BS and DP.*

Proof: In order to prove this lemma, we just use the fact that both policies are *non-idling* which means that whenever there is something ready to be executed, the resource is busy.

By definition of the resource busy functions, we have:

$$W^{BS}(t) = \sum_{(k,n)} C_{k,n}\mathbf{1}_{\{A_{k,n}<t\}} - \int_0^t \beta^{BS}(u)du, \tag{6}$$

$$W^{DP}(t) = \sum_{(k,n)} C_{k,n}\mathbf{1}_{\{A_{k,n}<t\}} - \int_0^t \beta^{DP}(u)du. \tag{7}$$

Therefore, if $\beta^{BS}(t) = \beta^{DP}(t)$ for all t, then the workloads will coincide. Let us assume that the busy functions are not the same under both policies. By the right continuity of the functions $\beta^{DP}(t)$ and $\beta^{BS}(t)$, there exists a time $t_0 > 0$ such that:

$$\begin{aligned} \beta^{DP}(t) &= \beta^{BS}(t), \quad \forall t \ s.t.\ 0 \leq t < t_0, \\ \beta^{DP}(t_0) &\neq \beta^{BS}(t_0). \end{aligned}$$

Let us assume that we have $\beta^{DP}(t_0) = 0$ and $\beta^{BS}(t_0) = 1$. Since the release times of all tasks as well as their execution durations are the same under both policies, and using the fact that the busy functions are the same in both cases up to time t_0, Equations (6) and (7) yield

$$W^{BS}(t_0) = W^{DP}(t_0).$$

Now, using the non-idleness of policy DP, $\beta^{DP}(t_0) = 0$ means that $W^{DP}(t_0) = 0$. As for BS, we have $\beta^{BS}(t_0) = 1$ and $W^{BS}(t_0) = 0$ which is only possible if there is an arrival at time t_0 in BS. But since the arrival times coincide under BS and DP, this implies that there is an arrival under DP at time t_0 as well, contradicting the fact that $\beta^{DP}(t_0) = 0$. The same argument holds for proving that $\beta^{DP}(t_0) = 1$ and $\beta^{BS}(t_0) = 0$ is impossible. ■

Definition 1. [busy period, cluster] A busy period $\mathcal{B}$ is a time interval $[t_1, t_2[$ such that $\beta(t) = 1$ for all $t \in [t_1, t_2)$ and $W(t_1) = 0$, $W(t_2^+) = 0$ (where t^+ is the limit $\lim_{y\downarrow t} y$). The set of all instances arriving in a busy period, forms a cluster, and will be denoted by $\mathcal{C}(\mathcal{B})$ ou simply $\mathcal{C}$ when the busy period itself is not important.

An immediate corollary of Lemma 2 is that busy periods and clusters coincide under BS and DP.

The rest of this section is devoted to the proof of the following result.

THEOREM 1
i- If the SRT traffic is executed in the FIFO order under both DP and BS, then for each instance $\tau_{i,n}$ of the SRT traffic, $E_{i,n}^{DP} \leq E_{i,n}^{BS}$.
ii- If the SRT traffic is not necessarily FIFO but if all SRT instances are of the same size, then DP performs better than BS on the SRT traffic on average: for all cluster $\mathcal{C}$ of size $|\mathcal{C}|$,

$$\frac{1}{|\mathcal{C}|} \sum_{\tau_i \in \mathcal{S},\, \tau_{i,n} \in \mathcal{C}} E_{i,n}^{DP} \leq \frac{1}{|\mathcal{C}|} \sum_{\tau_i \in \mathcal{S},\, \tau_{i,n} \in \mathcal{C}} E_{i,n}^{BS}.$$

However, there exists some configurations where BS performs better than DP on some SRT instances.
iii- If the SRT traffic is executed in an arbitrary order and the size of the tasks are not all equal then there exists some configurations where BS performs better than DP on average.

Before going on with the proof, this theorem calls for several comments.

- First note that Parts *i* and *ii* essentially give general positive results: under such and such conditions DP performs than BS for all possible sets of tasks. On the other hand, Part *iii* only says that there exist well chosen cases where BS is better than DP. The experiments presented in Section 5 will add some insight on this by showing that those cases are somehow rare. However, they cannot be dismissed.

- In Part *i*, we assume that the SRT traffic is executed in the FIFO order under both DP and BS. This does not mean that the FIFO order is imposed in a static way by the priorities, but only that the execution of the tasks happens to be FIFO, which is a weaker requirement.

The three parts (*i,ii,iii*) will be proved respectively in the three following subsections.

3.2. Part i- The FIFO Case

In this section, all instances of SRT tasks are executed in the FIFO order. With no loss of generality, we can assume that there exists a single SRT task τ_m, with $m = p+1$, that gathers all the instances of SRT tasks.
The first step is to compare DP and BS over a single busy period involving a cluster $\mathcal{C}$ (introduced in Definition 1). The global comparison will be derived from the individual comparisons over each cluster of the trajectory.
Over one busy period, we transform DP into a fixed non-preemptive priority policy, called EP in the following, which behaves exactly as DP by choosing appropriate priorities. Consider the completion times of all the instances in the cluster $\mathcal{C}$ under DP and assign priorities to all the instances according to the order of their completion time. Namely, $\pi^{EP}(k,n) = \#\{(i,j) \in \mathcal{C} : E^{DP}_{i,j} < E^{DP}_{k,n}\}$.

LEMMA 3 *For all instances $\tau_{k,n}$ in $\mathcal{C}$, $E^{DP}_{k,n} = E^{EP}_{k,n}$.*

Proof: First note that EP is non-idling and has the same clusters and busy periods as DP. The proof holds by induction on the size of the cluster considered. If the cluster is made by only one task, then clearly, $E^{DP}_{k,n} = A_{k,n} + C_{k,n} = E^{EP}_{k,n}$. Now assume that the cluster is made of i instances. We remove the task executed last under DP, say instance $\tau_{a,b}$. We get a cluster made of $i-1$ tasks. By induction, $E^{DP}_{k,n} = E^{EP}_{k,n}$ for all tasks in this reduced cluster.
Under DP, the completions of all tasks in the original cluster, are the same as in the reduced cluster since $\tau_{a,b}$ is executed last. As for EP, task $\tau_{a,b}$ has the lowest priority by construction of

EP. Since the remaining $i-1$ tasks form a cluster, the additional task $\tau_{a,b}$ with the lowest priority is transmitted last. ■

LEMMA 4 *Consider an arbitrary instance of a SRT instance, say $\tau_{k,n}$. Then $E^{EP}_{k,n} \leq E^{BS}_{k,n}$.*

Proof: We first show that under EP the priorities of all SRT instances are ordered according to their arrival. Let $\tau_{k,n}$ be an arbitrary SRT instance. Under policy DP, the priorities of SRT tasks are static and FIFO (by hypothesis). Since $A_{k,n} < A_{k,n+1}$, $\pi^{DP}(k,n,t) < \pi^{DP}(k,n+1,t)$ for all t. Using the execution rule of policy DP, if instance $\tau_{k,n}$ has not yet been transmitted at time $A_{k,n+1}$, then both tasks are pending. Whenever there is an execution opportunity, $\tau_{k,n}$ has priority over $\tau_{k,n+1}$ by the FIFO rule, therefore, it will be transmitted earlier. This means that $E^{DP}_{k,n} < E^{DP}_{k,n+1}$. As for EP, by definition, $\pi^{EP}(k,n) < \pi^{EP}(k,n+1)$.
The priority order among SRT instances is the same under EP and BS. As for real-time instances, their priority in EP is a modification with respect to those in BS. By applying Lemma 14, given in Appendix A, we obtain $E^{EP}_{k,n} \leq E^{BS}_{k,n}$ for all n. ■

Now, the proof of Part i of Theorem 1 is a direct consequence of Lemmas 3 and 4.

3.3. Part ii- The equal size case

In this subsection, the sizes of all SRT instances are all equal to C. As first step, we consider the DP policy (the BS case will be treated in Lemma 7). The sequence of the end of execution of all the instances defines some total order γ on all the instances. Namely, $\gamma(k,n) < \gamma(k',n')$ if instance $\tau_{k,n}$ is executed before instance $\tau_{k',n'}$.

We want to compare the performance of two different executions of DP on the same set of tasks where only the priorities within the set of SRT tasks have been changed with the constraint that they must stay in the priority zone of SRT tasks : each HRT instance in its critical interval keeps a higher priority than all SRT instances and a HRT instance out of its critical interval has a lower priority than all SRT instances. These changes in the priorities induce changes on the order of execution of the tasks, namely γ.

Let $\beta_S(t)$ (resp. $\beta_H(t)$) be the right-continuous SRT indicator function (resp. HRT indicator function) defined as follows. If the processor works on an SRT (resp. HRT) instance then $\beta_S(t) = 1$ (resp. $\beta_H(t) = 1$) and $\beta_S(t) = 0$ (resp. $\beta_H(t) = 0$) otherwise. In other words $\beta_S(t)$ tells us if the processor works on the SRT traffic at time t or not.

In the following, we will compare two orders, γ and γ', induced by some changes in the priorities on the SRT tasks. Since $\beta_S(t) + \beta_H(t) = \beta(t)$ is the total busy indicator, under the orders γ and γ', we have $\beta_S(t) + \beta_H(t) = \beta'_S(t) + \beta'_H(t)$ because in both cases the policy (DP here) is non-idle (see Lemma 2).

Lemma 5 *We consider two executions under DP, one of them following the order γ and the other one following γ'. If the SRT instances are all of the same size, then the SRT indicators are the same:* $\forall t \geq 0, \quad \beta_S(t) = \beta'_S(t)$.

Proof: We consider two executions, one of them following the order γ and the other one, γ'. Since $\beta_S(t)$ and $\beta'_S(t)$ are right-continuous functions, there exists t_0 which is the smallest time where $\beta_S(t_0) \neq \beta'_S(t_0)$ and $\beta_S(t) = \beta'_S(t)$ for all $t < t_0$. This means that one of them (say $\beta_S(t_0)$) changes while the other one ($\beta'_S(t_0)$) remains the same. Let us first consider the case where $\beta_S(t_0)$ changes from 1 to 0 while $\beta'_S(t_0)$ remains equal to 1 with $\beta_S(t_0) + \beta_H(t_0) = \beta'_S(t_0) + \beta'_H(t_0) = 1$

Since both SRT indicators coincide up to time t_0 and since the orders γ and γ' agree on all HRT instances, the set of HRT instances pending at time t_0 are the same under both executions. Also, the amount of SRT work pending at time t_0 is the same under both executions (and is positive because $\beta'_S(t_0) = 1$). Since the size of SRT tasks are equal, this means that the same number of SRT instances have been completed under γ and under γ'. If β_S changes at t_0, this means a SRT instance just completed its execution. Since the same number of SRT instances have been completed up to t_0 under γ', a SRT instance also just completed its execution under γ'.

At time t_0 the situation is the following.
(A) There exists a pending HRT instance $\tau_{k,n}$ such that $\gamma(k,n) < \gamma(h,u)$ for all pending SRT instances $\tau_{h,u}$.
(B) On the other hand, in the second execution, there exists a pending SRT instance $\tau_{i,j}$ such that $\gamma'(i,j) < \gamma'(a,b)$ for all pending HRT instances $\tau_{a,b}$. Property (A) means that one HRT instance, $\tau_{k,n}$ is in its critical interval while property (B) says that all HRT instances are in the non-critical mode. This is impossible so that t_0 does not exists.

The case where $\beta_S(t_0)$ changes from 0 to 1 is symmetrical with situations (A) and (B) reversed. A HRT task has just completed its execution under both γ and γ' and under γ a SRT gets the resource while under γ' a HRT instance gets the resource. This is also impossible.

■

Lemma 6 *Under DP, if all SRT instances have the same size C then the average completion times of SRT instances under priorities γ and γ' are the same: for each cluster $\mathcal{C}$,*

$$\sum_{\tau_i \in \mathcal{S},\, \tau_{i,n} \in \mathcal{C}} E_{i,n} = \sum_{\tau_i \in \mathcal{S},\, \tau_{i,n} \in \mathcal{C}} E'_{i,n}.$$

Proof: If the busy period associated with cluster $\mathcal{C}$ starts at time h, If we consider the departure times of all the SRT instances under γ, then the first departure occurs at time $t_1 = \inf\{t | \int_h^t \beta_s(u)du \geq C\}$. The second departures occurs at time $t_2 = \inf\{t | \int_h^t \beta_s(u)du \geq 2C\}$. More generally, the k^{th} departure occurs at time $t_k = \inf\{t | \int_h^t \beta_s(u)du \geq kC\}$. Since the SRT busy indicators β_s and β'_s coincide, the instants of the departures are the same under γ and under γ'

(even if the departing instances are different). Therefore the sum over all instances in the cluster is the same under both orders. ■

If we consider policy BS, then a similar lemma can be shown.

LEMMA 7 *Under BS, if all SRT instances have the same size C then the average completion times of SRT instances under priorities γ and γ' are the same: for each cluster $\mathcal{C}$,*

$$\sum_{\tau_i \in \mathcal{S},\, \tau_{i,n} \in \mathcal{C}} E_{i,n} = \sum_{\tau_i \in \mathcal{S},\, \tau_{i,n} \in \mathcal{C}} E'_{i,n}.$$

Proof: The same technique as for Lemmas 5 and 6 can be used. ■

Proof: (Part *ii* of Theorem 1). Let γ' be the original order of execution for DP and let γ'' be the original order of execution for BS. We can construct γ such that all SRT tasks are executed under the FIFO order under BS as well as DP. For example, choose $\pi(k,n) = (m, A_{k,n})$ (this puts all SRT instances in a single task and orders them according to the date of arrival). Now, we know that under γ, DP has the same average performance as under γ' (Lemma 6). Under γ, BS has the same average performance as under γ'' (Lemma 7). Finally DP performs better than BS under γ for each SRT instance (Part *i* of Theorem 1) and *a fortiori* on average.

The second assertion about the inversion on some instances is proved via the construction of an example similar to the construction presented in the forthcoming Figure 2, with two SRT tasks with the same size and $E_{3,1}^{BS} < E_{3,1}^{DP}$. ■

3.4. Part iii- An example where BS performs better than DP

In this section, we construct a configuration where the response times of some SRT tasks are smaller under BS than under DP. Let us consider the trajectory illustrated in Figure 2 with 2 instances of HRT tasks ($A_{1,1} = 0$ with $C_1 = 5$, $A_{2,1} = 3$ with $C_2 = 2$) and 2 instances of SRT tasks ($A_{3,1} = 6$ with $C_3 = 2$, $A_{4,1} = 4$ with $C_4 = 7$). Considering that neither instance $\tau_{1,1}$ nor instance $\tau_{2,1}$ are in a critical interval, we obtain on this trajectory $E_{1,1}^{BS} = 5$, $E_{2,1}^{BS} = 7$, $E_{3,1}^{BS} = 9$, $E_{4,1}^{BS} = 16$ for BS and $E_{1,1}^{DP} = 5$, $E_{2,1}^{DP} = 16$, $E_{3,1}^{DP} = 14$, $E_{4,1}^{DP} = 12$ for DP.
Note that for $\tau_{3,1}$, BS performs better than DP because $E_{3,1}^{BS} = 9 < E_{3,1}^{DP} = 14$. This inversion is due to the fact that SRT tasks are executed in a non FIFO order. In this example, SRT tasks are not of the same size, it is thus possible that BS performs better than DP on average. This is the case here since the average response time under DP is $1/2 \cdot (E_{3,1}^{BS} - A_{3,1} + E_{4,1}^{BS} - A_{4,1}) = 8$ while under BS it is equal to 7.5 .

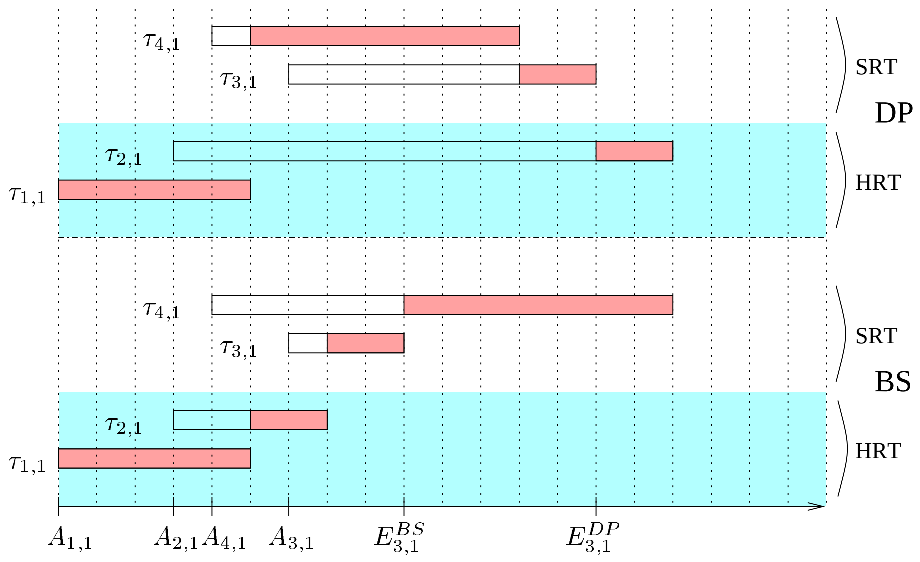

Figure 2. A trajectory under DP and BS in the non-preemptive case, where BS performs better than DP for $\tau_{3,1}$.

4. Dual-Priority VS Background Scheduling : the preemptive case

Several simulations published in [9, 11, 3, 4] suggest to us that DP improves greatly the response time of SRT tasks over BS in the preemptive case as well. However, as for the non-preemptive case, to the best of our knowledge, no formal comparison between DP and BS has ever been proposed. In the preemptive case, the priorities defined for BS and DP respectively are the same as in Section 3 (non-preemptive case). The only difference lies in the resource allocation rule which is now, for both policies :

the instance of the task being executed at time t is the pending task of highest priority.

The difference is that the task being executed can change suddenly, (for example as soon as a high priority task is released) and not only at completion times as in the non-preemptive case. Thus, the execution of a task $\tau_{k,n}$ can be split in several *pieces*, $\tau_{k,n,i}$ being the ith piece with execution beginning and completion respectively denoted by $B_{k,n,i}$ and $E_{k,n,i}$. Each interruption in the processing of a task is caused by an instance of a higher priority task that takes the resource. Note that under DP, the change can occur when an instance entering a critical interval, has its priority promoted.

As for the non-preemptive case, we will compare DP and BS over each trajectory. Here again, DP performs always better than BS when all SRT tasks are FIFO. Surprisingly, the proof does not work in the general case for completely different reasons. In the non-preemptive case, changing the priorities only improves the lowest priority task (Lemma 14), while in the preemptive case, all tasks with an improved priority benefit from it (Lemma 15). The problem for the preemptive

case comes from the comparison of BS with a fixed priority policy that behaves as DP, where the FIFO property is needed to have compatible priorities.
As for the non-preemptive case, feasibility under BS is always equivalent to $\mathbf{R}_k^{BS} \leq D_k \quad \forall \tau_k \in \mathcal{P}$. The computation of $\mathbf{R}_k^{BS}$ in the preemptive case is classical, it has been for instance studied by Joseph and Pandya [16] and later by Audsley et al. [2] :

$$\mathbf{R}_k^{BS} = I_k \tag{8}$$

where I_k, the longest time that all higher priority tasks can occupy the resource, is the limit of the sequence

$$I_k^0 = 0, \quad I_k^n = C_k + \sum_{\forall j<k} \left\lceil \frac{I_k^{n-1}}{T_j} \right\rceil C_j \tag{9}$$

I_k is computed starting with $I_k^0 = 0$ until convergence or until $I_k^n > D_k - C_k$.

4.1. Busy periods

LEMMA 8 *The total workload is the same under BS and DP.*

Proof: The proof is exactly the same as in the non-preemptive case, see Lemma 2. ■

An immediate corollary is that busy periods as well as clusters are the same under DP and under BS. One can also verify that the workload (as well as busy periods and clusters) are the same under preemptive and non-preemptive systems because of the non-idleness.

In the rest of this section, we will prove the following theorem

THEOREM 2
i- If the SRT tasks are executed in the FIFO order under both DP and BS, then for each SRT instance $\tau_{k,n}$, $E_{k,n}^{DP} \leq E_{k,n}^{BS}$.
ii- If the SRT tasks are not FIFO, then there exists configurations where BS performs better than DP on average.

Note that Part *i* of Theorem 2 is similar to Part *i* of Theorem 1. However, unlike in the non-preemptive case, BS may perform better than DP on average as soon as the FIFO condition is not satisfied, even when the SRT instances are all of the same size. This will be illustrated in Section 4.3.

4.2. Part i-The FIFO case

In a similar way as in the non-preemptive case, we construct an intermediate policy EP with fixed priorities that behaves exactly as policy DP. However, the construction is more involved.
To construct EP, we consider each piece $\tau_{k,n,i}$ of execution of task $\tau_{k,n}$ under DP as an instance of task τ_k, called $\tau_{k,n,i}$, with arrival time $A_{k,n}$. Therefore, under EP, several instances (as many

as there are pieces under DP for $\tau_{k,n}$) are released at the same instant. The priorities under EP are given according to the end of execution of all the pieces:

$$\pi^{EP}(k,n,i) = \#\{(x,y,z)|E^{DP}_{x,y,z} < E^{DP}_{k,n,i}\}. \tag{10}$$

Lemma 9 *For all k, n, i, $E^{DP}_{k,n,i} = E^{EP}_{k,n,i}$.*

Proof: The proof holds by induction on the size of the cluster that includes $\tau_{k,n}$. If $\tau_{k,n}$ is alone in the cluster, then $\tau_{k,n}$ is not interrupted during its execution and it is formed by a single piece $\tau_{k,n,1} = \tau_{k,n}$. By Lemma 8, we have $E^{DP}_{k,n} = E^{EP}_{k,n,1}$.
Now, let us assume that the cluster is formed of n pieces. The reduced set composed of the same pieces except the last one to be executed under DP forms a cluster with $n-1$ pieces. By induction, the completion times are the same under both policies. As for the last task, it is executed last under DP, therefore, according to Equation (10) it has the lowest priority under EP and will be thus executed last among the considered cluster. ■

In the rest of this section, we consider that all SRT instances are ordered in the FIFO order under BS. As for the non-preemptive case, we can assume that there exists a single SRT task τ_m, with $m = p+1$, that gathers all the instances of SRT tasks.
We first need to modify the tasks under BS by considering each piece of execution $\tau_{k,n,i}$ (defined in Section 4.2 to be a new task with arrival time $A_{k,n}$ and priority $\pi^{BS} = (k,n,i)$.

Lemma 10 *Under EP, the SRT tasks verify $\pi^{EP}_{m,n,i} < \pi^{EP}_{m,n+1,j}$ and $\pi^{EP}_{m,n,i} < \pi^{EP}_{m,n,i+1}$ for all n, j, i.*

Proof: By Lemma 9, we have $E^{DP}_{m,n,i} = E^{EP}_{m,n,i}$. By construction of the tasks $\tau_{m,n,i}$, we have $E^{DP}_{m,n,i} < E^{DP}_{m,n+1,j}$ and $E^{DP}_{m,n,i} < E^{DP}_{m,n,i+1}$ for all n, j, i. By definition of the priorities under EP, see Equation (10), we obtain $\pi^{EP}_{m,n,i} < \pi^{EP}_{m,n+1,j}$ and $\pi^{EP}_{m,n,i} < \pi^{EP}_{m,n,i+1}$ for all n, j, i. ■

Lemma 11 *Under the FIFO assumption on SRT tasks $E^{DP}_{m,n,i} \leq E^{EP}_{m,n,i}$ for all n and i.*

Proof: First note that a task $\tau_{k,n,i}$ may not be executed in a single piece under BS, unlike under EP. If a single cluster is considered, then all the SRT tasks have compatible priorities under BS and EP using Lemma 10. As for the HRT tasks, their priorities can be very different under both policies. However, for each SRT task $\tau_{m,n,i}$, all HRT tasks have higher priority than $\tau_{m,n,i}$ under BS. Combining the two previous properties, the set of tasks with higher priority under EP is included in the set of tasks with higher priority under BS. Therefore we can apply Lemma 15, given in Appendix B, which implies that $E^{EP}_{m,n,i} \leq E^{BS}_{m,n,i}$. ■

Lemma 12 *For any task $\tau_{k,n}$, $E^{DP}_{k,n} \leq E^{BS}_{k,n}$.*

Proof: First, note that $E^{DP}_{k,n} = E^{DP}_{k,n,h}$, where piece $\tau_{k,n,h}$ is the last piece of task $\tau_{k,n}$ under DP. Using Lemma 9, we obtain $E^{EP}_{k,n,h} = E^{DP}_{k,n,h}$. Now, using Lemma 11, $E^{EP}_{k,n,h} \leq E^{BS}_{k,n,h}$. Finally, using Lemma 10, $E^{BS}_{k,n} = E^{BS}_{k,n,h}$, which ends the proof. ■

4.3. Part ii- An example where BS performs better than DP

We now show an example of a trajectory for which the preemptive scheduling under DP is not better than under BS for SRT tasks on average. As for the non-preemptive case of Section 3.4, this counter example uses SRT tasks arriving in a non FIFO order. Here, we also need a HRT task which has its priority promoted.

Let us consider a trajectory (see Figure 3) with a single HRT task ($A_{1,1} = 3$ and $C_1 = 3$) and two SRT tasks ($A_{2,1} = 6$ and $C_2 = 7$, $A_{3,1} = 0$ and $C_3 = 7$), the HRT instance $\tau_{2,1}$ entering its critical interval at time 9. Under this trajectory, we obtain $E^{DP}_{1,1} = 10$, $E^{DP}_{2,1} = 16$, $E^{DP}_{3,1} = 17$ and $E^{BS}_{1,1} = 6$, $E^{BS}_{2,1} = 13$, $E^{BS}_{3,1} = 17$.

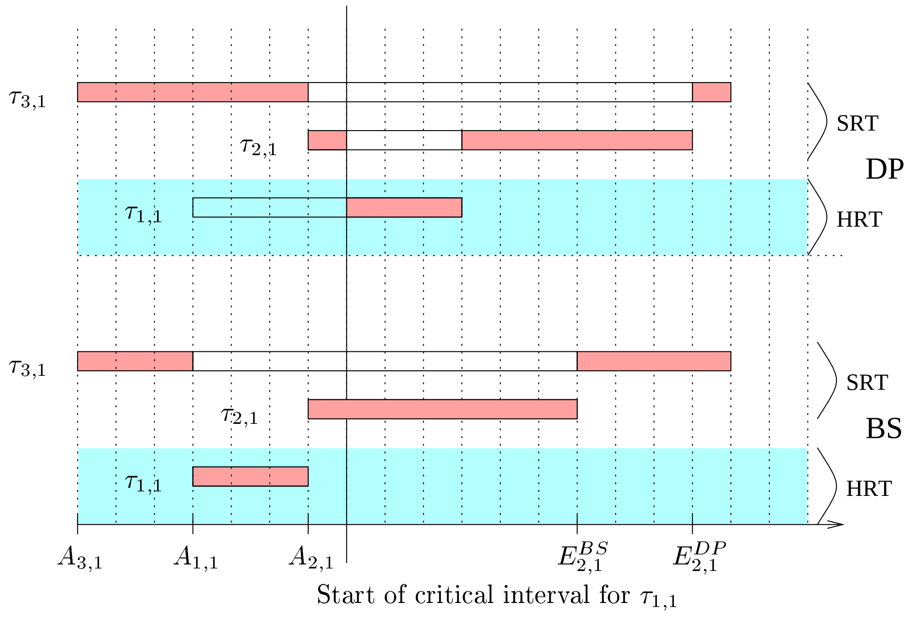

Figure 3. A trajectory under DP and BS in the preemptive case, where BS performs better than DP for $\tau_{2,1}$.

Figure 3 represents this trajectory, one notes that $E^{BS}_{2,1} < E^{DP}_{2,1}$, thus for $\tau_{2,1}$ BS performs better than DP. Again, this phenomenon is due to the non-FIFO feature of the SRT traffic. Note that it is possible to build examples for which the difference $E^{DP}_{m,j} - E^{BS}_{m,j}$ is arbitrarily large, for some SRT instances $\tau_{m,j}$. Since the scheduling is here preemptive, BS may outperform DP even if the SRT tasks of the example are of the same size. This is actually the case since the average response time under DP is 13.5 versus 12 for BS.

5. Experiments

Many simulation results have been published in the literature that show a substantial gain in terms of response time for SRT traffic when DP was used instead of BS. Under heavy load (> 70%), the gains usually encountered were above 60% (see experiments 5.3 and 5.4 in [11], Figure 4 in [4], Figure 2 in [23] and Figure 5 in [13]). In this section, in order to exhibit the practical implications of Theorem 12, we present new simulations done in the non-preemptive case.

In the first experiment, we consider a realistic CAN-based in-vehicle application provided by the car industry (see [24] for a detailed description) where 6 devices (e.g. engine controller, automatic gear box, ...) exchange messages. The traffic consists of a set of 12 HRT messages (e.g. speed and torque from the engine controller) with periods $T_1 = 10$, $T_2 = 14$, $T_3 = 20$, $T_4 = 15$, $T_5 = 20$, $T_6 = 40$, $T_7 = 15$, $T_8 = 50$, $T_9 = 20$, $T_{10} = 100$, $T_{11} = 50$ and $T_{12} = 100$ms, all having a size of 125 bits. The periodic sources are assumed to begin transmitting from the start of the simulation. In addition, there exists a stream of SRT frames whose inter-arrival times are exponentially distributed. Among the SRT traffic, we distinguish 15 different messages all of size 100 bits. The transmission rate of the CAN bus is 125kbit/s and the total network load induced by the 27 messages is 90% with 53.46% for the periodic part.

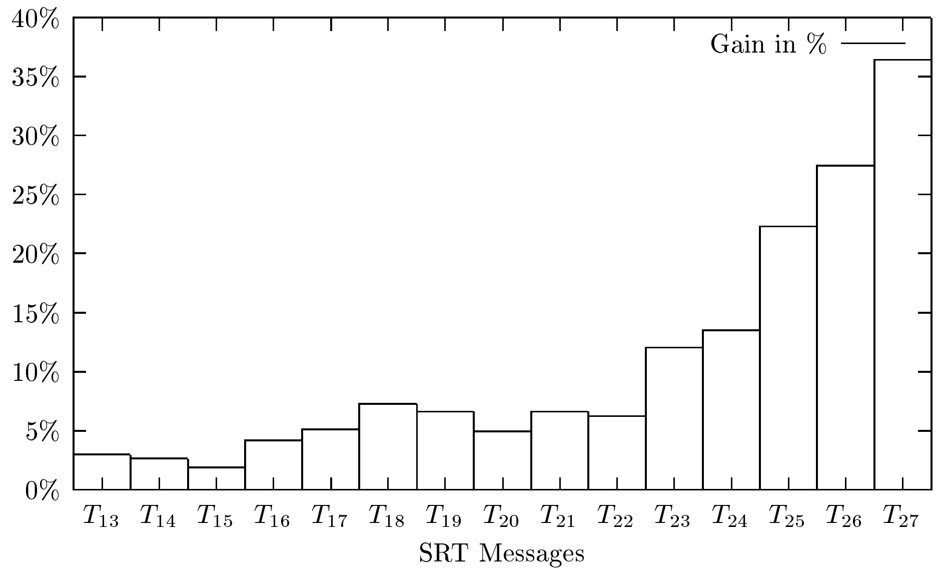

Figure 4. Average gain in response time of DP over BS for each SRT message with 12 HRT messages.

In order to show the importance of the priority order among the SRT messages, we have imposed the SRT messages to be emitted in a quasi-LIFO order, that is in the order :
$\tau_{27,1}, \tau_{26,1}, \tau_{25,1}, \cdots, \tau_{15,1}, \tau_{14,1}, \tau_{13,1}, \tau_{27,2}, \cdots$, the inter-arrival time between two consecutive instances being exponentially distributed. The simulations using DP and BS policies respectively were run on more than 10,000 instances of HRT messages.

The results, represented in Figure 4, show the gain (given in percentage) of DP over BS for each SRT message in terms of the average response time. It is worth noting that while the gain remains substantial for message τ_{27} (around 40%), this is not the case for messages in the high and mid-priority ranges (the gain for τ_{15} is for instance around 2%). This is very small compared to other simulations which were published in the literature.

In the second set of experiments, we have chosen an environment which was likely to exhibit the behaviour illustrated in Figure 2, where some SRT tasks (here messages) have a better response time under BS than under DP. We simulate a CAN priority bus with a stream composed of 15 different SRT messages of size 100 bits and with a single HRT message of length 115bits and period $\tau_1 = 1$ms. The transmission rate is 250kbit/s, the HRT message induces a load of 46% while the total load is 95%. Again, the SRT messages are released in a quasi LIFO order

$$\tau_{16,1}, \tau_{15,1}, \tau_{14,1}, \cdots, \tau_{4,1}, \tau_{3,1}, \tau_{2,1}, \tau_{16,2}, \cdots$$

The results of these simulations are shown in Figure 5. Here, it is remarkable to notice that 4

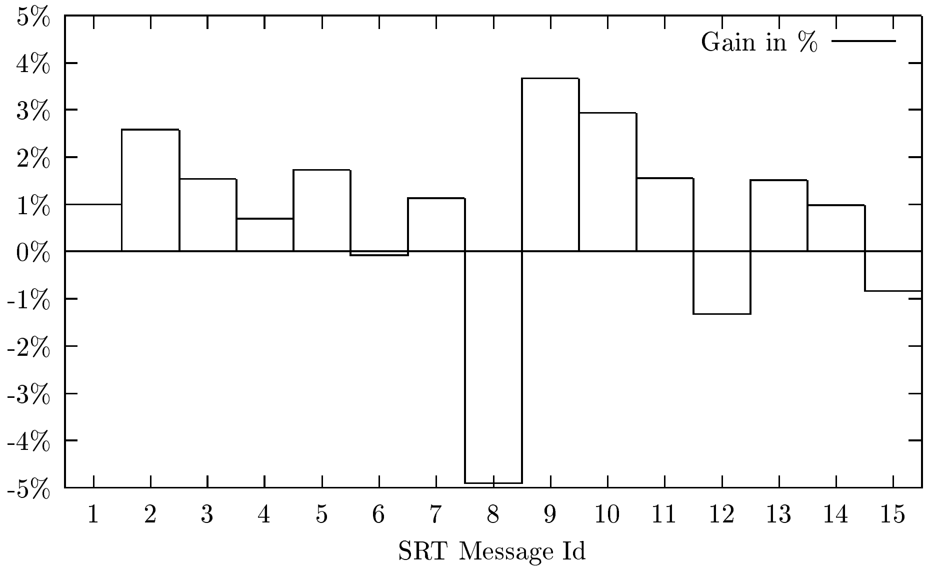

Figure 5. Average gain in response time of DP over BS for each SRT message with 1 HRT message.

out of 15 SRT messages have an average response time better under BS than under DP, namely tasks $\tau_6, \tau_8, \tau_{12}$ and τ_{15}. Note also that the gain obtained for task τ_8 under BS is about 5%, and is the largest absolute gain. In our experiments, the inter-arrival times for each SRT message is a sum of 15 iid exponential variables and therefore has a small variance. The global behaviour is thus "close" to a periodic case where the response times among SRT messages are known to vary drastically. This gives an explanation to the very different performances of messages 8 and 9.

We have also tried to find configurations where BS outperforms DP on average with stochastic arrival times for SRT traffic. None of the simulation experiments could show this result. In particular, starting with the example of Figure 2 and introducing a very small variability of the arrival times of SRT messages (uniform distribution on a small interval) changes the relative performance of DP and BS and DP happens to beat BS on average even when the size of the interval is very small with respect to the period (a ratio down to 10^{-2} in our experiments).

6. Implementation Issues

Throughout the paper we have seen that the FIFO ordering for SRT tasks is critical for the efficiency of DP when all SRT messages have the same size. In this section, we are concerned with the practical problem of ensuring the FIFO ordering for SRT traffic.

In a centralised framework, the FIFO ordering for SRT tasks is easily achieved through prioritization. The problem that will be addressed in this section is to find a way of imposing the FIFO ordering for the scheduling of messages in a distributed environment. We will focus on the CAN priority bus but the principles of the analysis remains valid for other priority buses such as the J1850 [28] or VAN [12]. The use of DP for message scheduling on CAN has been proposed by Tindell and Hansson in [30]. The mechanisms described in the following are compatible with the use of DP on CAN.

Each station may emit SRT messages as well as HRT messages. Each message is composed of K bits for priority encoding and M bits for control and data. In the following, we will propose a way to encode the FIFO feature on those K bits, provided that all stations have a synchronous clock. Note that the encoding of information using some bits of the identifier is rather classical in CAN, see for instance [31] and [25]. To synchronise the stations, we have to add an initialisation phase, which consists in sending a special frame that serves as global starting signal. This signal will serve as the origin of time, the granularity of time being the bit-time. Since the CAN protocol ensures that all nodes are synchronised on the bit-time, all stations will have coherent clocks.

For HRT messages, the K priority bits are used in the following way : the first 2 bits are set to 00 when the message is in a critical interval and 10 otherwise (see Figure 6). The following $K-2$ bits are used to encode the priority level. This is possible as long as $K-2 \geq \log_2(p)$ where p is the number of HRT frames. For SRT messages, the first 2 bits of the K priority bits are set to 01. The following $K-S-2$ bits are used to encode the current time t, common to all stations by assumption, which is called the *time-stamp* of the frame. The last $S = \lceil \log_2(\# \text{ stations }) \rceil$ bits are used to encode the identifier of the sender station (in order to break ties in case of identical time-stamps). There is enough space as long as $K-S-2 \geq \log_2(t)$. Note that distinguishing between the 2 types of HRT frames and SRT frames using the first 2 bits of the CAN frame has already been proposed in [30].

With this method the priority for an instance of a SRT message is its release instant. Therefore, with the assumption that all stations are synchronised, it makes sure that all the SRT traffic satisfies the FIFO ordering. However, the drawback of this method is the fact that it is impossible to encode SRT messages after time 2^{K-S-2}. In order to deal with this problem, we propose to add one particular SRT message (called the *reset message*, denoted τ_{m+1}) that will be released

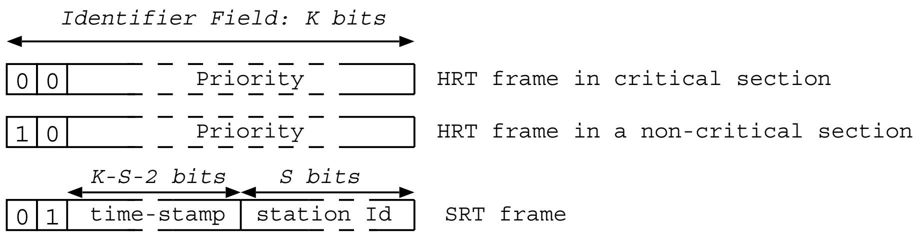
Identifier Field: K bits
0 0 Priority
HRT frame in critical section
1 0 Priority
HRT frame in a non-critical section
K-S-2 bits
S bits
0 1 time-stamp station Id
SRT frame

Figure 6. Identifier field of the CAN frame for HRT frames and SRT frames.

every H units of time, with a priority level defined as for the SRT messages (i.e. function of its time-stamp and station identifier). On reception of this particular message, each station decreases its time clock by H units of time, as well as the time-stamp of all SRT messages which are waiting to be sent.

The question now is how to choose H and K such that no overflow of the time clock ever occurs. First, let us compute an upper bound L on the length of all busy periods in the system, this will give us a bound on the response time of the reset message. This calculus can be done under some assumptions on the SRT traffic, namely, a deterministic load arrival bound such as a (σ, ρ) condition (see [8]). We assume that for any message, τ_k, the work released in any interval of size t is bounded by the affine formula $\sigma_k + \rho_k t$. In particular, periodic and sporadic messages satisfy a (σ, ρ) condition. The deterministic stability of the system says: $\sum_{k=1}^{m+1} \rho_k < 1$. Now, let the interval $[t_1, t_2]$ be a busy period of the system. This means that the total load is null at time t_1 and at time t_2. Therefore, the length of the busy period equals the load which has arrived during the interval $[t_1, t_2]$. This yields

$$L \leq \frac{\sum_{k=1}^{m+1} \sigma_k}{1 - \sum_{k=1}^{m+1} \rho_k}.$$

Lemma 13 *With*

$$H = 2^{K-S-2} - L, \tag{11}$$

there is no time overflow.

Proof: The k^{th} instance of the reset message is released at kH and completes its execution at e_k, when the clock is set to zero, see Figure 7.

The initial reset message being released just after the synchronisation and before all other SRT instances, no overflow occurs until e_0. Suppose that with the chosen H no time overflow has occurred until e_{k-1}. Because of the FIFO order, SRT instances pending at e_{k-1} have been released after $(k-1)H$. Due to the reset at e_{k-1}, their time-stamps and those of the instances released in $[e_{k-1}, e_k[$ vary between 0 and $e_k - e_{k-1}$. To avoid overflow, $e_k - e_{k-1}$ must be shorter than the longest time that can be encoded, i.e. shorter than 2^{K-S-2}. Since all response times are shorter than the longest busy period, we have $e_k \leqslant kH + L$. Furthermore $e_{k-1} \geqslant (k-1)H$. Thus,

$$e_k - e_{k-1} \leqslant kH + L - (k-1)H = H + L = 2^{K-S-2}.$$

■

Actually any choice $H \leqslant 2^{K-S-2} - L$ is an acceptable solution, but to limit the overhead induced by the reset message, H should be chosen as long as possible.

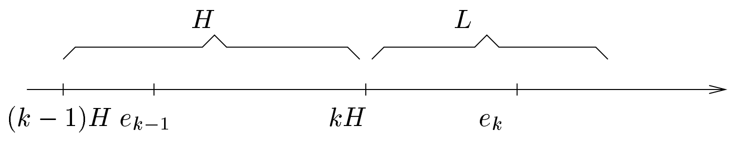

Figure 7. Choosing the period H of the reset message.

Note that the reset message contributes to the total load and also to the bound L of the busy periods, with σ_{m+1} being the transmission time of the reset message and $\rho_{m+1} = \sigma_{m+1}/H$. When K and the characteristics of the traffic are fixed, then Equation (11) can be solved in H (quadratic form).

In the context of CAN, the number K of bits allocated for priority encoding is fixed to either 11 bits (CAN 2.0A) or 29 bits (CAN 2.0B). A typical case (as for in-vehicle networks, see [27, 24]) would involve less than 64 stations, so that $S = 6$ is enough.

With $K = 11$, this leaves only 3 bits for time-stamp encoding. Considering the 12 HRT messages described in Section 5 and a SRT traffic inducing a load of 75% made of 20 messages of 100 bits, the total load at 500kbit/s is 86%. When solving (11), we obtain two negative values for the period H of the reset message. Thus the proposed method is not applicable on CAN2.0A.

With CAN2.0B, $K = 29$ thus 21 bits are left for the time-stamp encoding which enables to encode a time-stamp range of 4194 ms. Using the same traffic as previously, the bound L on the longest busy period is 76 ms and the solution of Equation (11) is $H = 4118$ ms. In these conditions, the load overhead of the reset message is 4.10^{-5} which is neglectable.

7. Conclusion

In this paper, it has been proven for the preemptive as well as for the non-preemptive scheduling that the response times of the SRT traffic are always better under the Dual-Priority than under Background-Scheduling if and only if the whole SRT traffic is FIFO. This was confirmed by simulations of a CAN bus which have shown the response times of some SRT messages to be smaller under BS than under DP when the FIFO condition is not satisfied. The result of the study reinforces the results hinted by previous experimental studies showing substantial gain of DP over BS by providing a theoretical basis which gives practical guidelines for application designers willing to implement DP.

As suggested by an anonymous reviewer, one could imagine a DP mechanism where the priority promotion time Y is not a priori set to $Y = A_{k,n} + D_k - \mathbf{R}_k$ but where it could take any value between $A_{k,n}$ and $A_{k,n} + D_k - \mathbf{R}_k$. For example, with $Y = A_{k,n}$, the resulting policy is BS. For each task set, there exists an optimal priority promotion time scheme (regarding SRT response times). The interesting open problem is now to find these optimal promotion times with respect to the characteristics of the task set.
Finally, such a path-wise method of comparing scheduling policies may also be used for several other non-idling scheduling policies such as EDL [7], PE [19] or EPE [29]. This is currently under investigation.
Acknowledgement The authors would like to thank Gerald Cabus for some helpful discussions about the implementation issues of the FIFO feature on priority buses as well as the anonymous reviewers who suggested to analyse the average case.

Appendix A

Changing priorities (non-preemptive case) This appendix is devoted to the proof of a general technical lemma about the effect on the change of priorities in the BS policy in the non-preemptive case.

A formula for the response time for an instance $\tau_{k,n}$ can be derived from [22] (p.22). It satisfies the following equality.

$$R_{k,n} = C_{k,n} + \min\{t \geq 0 \mid \Omega_{k,n}(A_{k,n}) + \Gamma_{k,n}(t) + \rho_{k,n}(A_{k,n}) = t\}, \tag{A.1}$$

where

- $\Omega_{k,n}(x)$ is the workload present at time x contributed by all instances with priority higher than $\pi(k,n)$,
- $\Gamma_{k,n}(x) = \sum_{(i,j)} \mathbf{1}_{\{\pi(i,j)<\pi(k,n)\}} \mathbf{1}_{\{A_{k,n} \leq A_{i,j} \leq A_{k,n}+x\}} C_{i,j}$ is the high priority work arrived between the release time, $A_{k,n}$ and time $A_{k,n} + x$.
- $\rho_{k,n}(x)$ is the remaining execution time of an instance with a priority lower than $\pi(k,n)$, which has started its execution before time x and is not completed yet.

As for the completion time,

$$\begin{aligned} E_{k,n} &= A_{k,n} + R_{k,n} \\ &= A_{k,n} + C_{k,n} + \min\{t \geq 0 \mid \Omega_{k,n}(A_{k,n}) + \Gamma_{k,n}(t) + \rho_{k,n}(A_{k,n}) = t\}. \end{aligned}$$

We construct a new fixed priority function π' in the following way: $\pi'(m,n) = \pi(m,n)$ for all instances of the lower priority task τ_m. As for any other task τ_k, $\pi'(k,i)$ is arbitrary (possibly larger than $\pi'(m,n)$).

All quantities related with the new priority will be denoted with a prime sign.

LEMMA 14 *For any instance of the lowest priority class, $\tau_{m,n}$, $E'_{m,n} \leq E_{m,n}$.*

Proof:

Consider an instance of the lower priority task $\tau_{m,n}$. This task belongs to a cluster, say $\mathcal{C}$ of all instances involved in a busy period $[t_1, t_2[$. In the following of the proof, we will only consider the instances of tasks belonging to $\mathcal{C}$, since no other task will influence the completion time of task $\tau_{m,n}$ under both priorities.

Under the original priorities, since $\tau_{m,n}$ is the instance with the lowest priority released so far, $\rho_{m,n}(A_{m,n}) = 0$ and

$$E_{m,n} = A_{m,n} + C_{m,n} + \min\left\{t \geq 0 \mid \Omega_{m,n}(A_{m,n}) + \Gamma_{m,n}(t) = t\right\}. \tag{A.2}$$

As for the value of $E'_{m,n}$,

$$E'_{m,n} = A_{m,n} + C_{m,n} + \min\left\{t \geq 0 \mid \Omega'_{m,n}(A_{m,n}) + \Gamma'_{m,n}(t) + \rho'_{m,n}(A_{m,n}) = t\right\}. \tag{A.3}$$

In order to compare both values, we will examine closely the values of the different terms involved in Equations (A.2) and (A.3).

Since $\tau_{m,n}$ is the lowest priority tasks which has ever been released by time $A_{m,n}$, then

$$\begin{aligned}
\Omega_{m,n}(A_{m,n}) &= W(A_{m,n}) && \text{(A.4)}\\
&= W'(A_{m,n}) && \text{(A.5)}\\
&\geq \Omega'_{m,n}(A_{m,n}) + \rho'_{m,n}(A_{m,n}), && \text{(A.6)}
\end{aligned}$$

where, Equation (A.4) comes from the fact that when instance $\tau_{m,n}$ is released, it has the lowest priority so far, Equation (A.5) from the fact that the workload of all non-idling policies are equal (see Lemma 2) and Equation (A.6) from the definition of Ω' and ρ' which are both distinct part of the workload, $\Omega'_{m,n}(A_{m,n})$ is the fraction of the workload at time $A_{m,n}$ due to instances of high priority and $\rho'_{m,n}(A_{m,n})$ is the fraction of the workload due to an instance with lower priority than $\tau_{m,n}$ being transmitted at time $A_{m,n}$.

In addition, for each time t,

$$\begin{aligned}
\Gamma_{m,n}(t) &= \sum_{(i,j)} C_{i,j} \mathbf{1}_{\{\pi(i,j)<\pi(m,n)\}} \mathbf{1}_{\{A_{m,n} \leq A_{i,j} \leq A_{m,n}+t\}}, && \text{(A.7)}\\
&\geq \sum_{(i,j)} C_{i,j} \mathbf{1}_{\{\pi'(i,j)<\pi'(m,n)\}} \mathbf{1}_{\{A_{m,n} \leq A_{i,j} \leq A_{m,n}+t\}} && \text{(A.8)}\\
&= \Gamma'_{m,n}(t). && \text{(A.9)}
\end{aligned}$$

Inequality (A.8) comes from the fact that the set of all instances $(\tau_{i,j})$ such that $\pi(i,j) > \pi(m,n)$ arriving after time $A_{m,n}$ is exactly the set $\{\tau_{m,i}, i > n\}$. It is included in the set of instances such that $\pi'(i,j) > \pi'(m,n)$ and arriving after time $A_{m,n}$, since π' does not modify the priorities among the task τ_m. The complementary sets are included in the reversed direction. For all (i,j),

$$\mathbf{1}_{\{\pi(i,j)<\pi(m,n)\}} \mathbf{1}_{\{A_{m,n} \leq A_{i,j} \leq A_{m,n}+t\}} \geq \mathbf{1}_{\{\pi'(i,j)<\pi'(m,n)\}} \mathbf{1}_{\{A_{m,n} \leq A_{i,j} \leq A_{m,n}+t\}}.$$

Combining both Equation (A.6) and (A.9) yields $E_{m,n} \geq E'_{m,n}$. ■

Appendix B

Changing priorities (preemptive case) In the preemptive case, one can prove a more powerful result than Lemma 14. Here the response time of any instance is improved whenever its priority is improved, while Lemma 14 only considers the lowest priority task.

The completion time of an instance $\tau_{k,n}$ is given by the formula:

$$E_{k,n} = A_{k,n} + \min\big\{t \geq 0 \mid \Omega_{k,n}(A_{k,n}) + \theta_{k,n}(t) + C_{k,n} = t\big\},$$

where

- $\Omega_{k,n}(x)$ is the workload present at time x contributed by all instances with priority higher than $\pi(k,n)$,
- $\theta_{k,n}(x) = \sum_{(i,j)} C_{i,j} \mathbf{1}_{\{\pi(i,j)<\pi(k,n)\{} \mathbf{1}_{\{A_{k,n} \leq A_{i,j} < x + A_{k,n}\}}$ is the high priority load arrived between the release time $A_{k,n}$ and time $A_{k,n} + x$.

We modify some priorities such that the priority of instance $\tau_{k,n}$ is improved: for all (i,j), $\pi'(i,j) < \pi'(k,n) \Rightarrow \pi(i,j) < \pi(k,n)$. Under this assumption on π', the following lemma can be established.

Lemma 15 $E_{k,n} \geq E'_{k,n}$

Proof: The proof is similar to the proof of Lemma 14.

Again, using the non-idling conservation Lemma 8, the busy periods as well as the clusters are identical under both priorities. Task $\tau_{k,n}$ belongs to one cluster involved in a busy period, $[t_1, t_2[$. In the following only tasks in this cluster will be considered.

$$E_{k,n} = A_{k,n} + \min\big\{t \geq 0 \mid \Omega_{k,n}(A_{k,n}) + \theta_{k,n}(t) + C_{k,n} = t\big\},$$

and

$$E'_{k,n} = A_{k,n} + \min\big\{t \geq 0 \mid \Omega'_{k,n}(A_{k,n}) + \theta'_{k,n}(t) + C_{k,n} = t\big\}.$$

A first sequence of inequalities gives

$$\begin{aligned} \Omega_{k,n}(A_{k,n}) &= W_{k,n}(A_{k,n}) && \text{(B.1)} \\ &\geq W'_{k,n}(A_{k,n}) && \text{(B.2)} \\ &= \Omega'_{k,n}(A_{k,n}), && \text{(B.3)} \end{aligned}$$

where $W_{k,n}(t)$ (resp. $W'_{k,n}(t)$) is the workload at time t of a system where all tasks with priority lower than $\pi(k,n)$ (resp. $\pi'(k,n)$)have been removed.

Equality (B.1) comes from the fact that in preemptive systems, low priority tasks have no influence on the execution of higher priority tasks. Equality (B.2) comes from the fact that the set of the instances with higher priority than $\tau_{k,n}$ under π includes the the set of the instances with higher priority than $\tau_{k,n}$ under π'. Equality (B.3) is the same as Equality (B.1) but with π'.

The second series of inequalities is

$$\begin{aligned}\theta_{m,n}(t) &= \sum_{(i,j)} C_{i,j}\mathbf{1}_{\{\pi(i,j)<\pi(m,n)\}}\mathbf{1}_{\{A_{m,n}\le A_{i,j}<A_{m,n}+t\}} && \text{(B.4)}\\ &\ge \sum_{(i,j)} C_{i,j}\mathbf{1}_{\{\pi'(i,j)<\pi'(m,n)\}}\mathbf{1}_{\{A_{m,n}\le A_{i,j}<A_{m,n}+t\}} && \text{(B.5)}\\ &= \theta'_{m,n}(t). && \text{(B.6)}\end{aligned}$$

Inequality (B.5) comes from the definition of π' which implies that for all (i,j),

$$\mathbf{1}_{\{\pi(i,j)<\pi(m,n)\}}\mathbf{1}_{\{A_{m,n}\le A_{i,j}<A_{m,n}+t\}} \le \mathbf{1}_{\{\pi'(i,j)<\pi'(m,n)\}}\mathbf{1}_{\{A_{i,j}A_{m,n}\le A_{i,j}<A_{m,n}+t\}}.$$

Combining both Equation (B.3) and (B.6) yields $E_{m,n} \ge E'_{m,n}$. ■

References

1. N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Engineering Journal*, 8(5):284–292, September 1993.
2. N.C. Audsley, A. Burns, M.F. Richardson, and A.J. Wellings. Hard real-time scheduling: The deadline monotonic approach. In *Proceedings 8th IEEE Workshop on Real-Time Operatings Systems and Software*, 1991.
3. G. Bernat. *Specification and Analysis of Weakly Hard Real-Time Systems.* Phd thesis, Universitat de les Illes Balears, Departament de Ciències Matemàtiques i Informàtica, January 1998.
4. G. Bernat and A. Burns. Combining (n, m)-hard deadlines and dual priority scheduling. In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, December 1997.
5. A. Burns, K. Tindell, and A.J. Wellings. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Transactions on Software Engineering*, 21(5):475–480, May 1995.
6. A. Burns and A.J. Wellings. Dual priority scheduling in ada95 and real-time posix. In *Proceedings of the 21st IFAC/IFIP Workshop on Real-Time Programming*, pages 45–50, 1996.
7. H. Chetto and M. Chetto. Some Results of the Earliest Deadline Scheduling Algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, October 1989.
8. R.L. Cruz. A calculus for network delay, part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):132–142, January 1991.
9. R. Davis. Dual Priority scheduling : A means of Providing Flexibility in Hard Real-Time Systems Systems. Technical report, Department of Computer Science, Univ. of York (UK), May 1994. Technical Report YCS230.

10. R. Davis, K. Tindell, and A. Burns. Scheduling slack time in fixed priority pre-emptive systems. In *Proceedings of the 14th IEEE Real-Time Systems Symposium*, pages 222–231, December 1993.

11. R. Davis and A.J. Wellings. Dual priority scheduling. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 100–109, December 1995.

12. Association Française de Normalisation (AFNOR). Véhicules routiers - transmission de données, 1990. document R13-708.

13. B. Gaujal and N. Navet. Traffic shaping in real-time distributed systems: a low-complexity approach. *Computer Communications*, 22(17):1562–1573, 1999. Also available as INRIA Research Report RR-3719.

14. L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive real-time uni-processor scheduling. Technical Report 2966, INRIA, september 1996.

15. International Standard Organization ISO. *Road Vehicles - Low Speed serial data communication - Part 2: Low Speed Controller Area Network*. ISO, 1994. ISO 11519-2.

16. M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

17. S. Lefebvre-Barbaroux, A. Jean-Marie, and C. Chaouiya. Problèmes d'ordonnancement d'une superposition de flux periodiques sous contraintes temps-réel. Rapport de Recherche 1576, INRIA, January 1992.

18. J. Lehoczky and S. Ramos-Thuel. Aperiodic task scheduling for hard-real-time systems. In *Proceedings IEEE Real-Time Systems*, pages 110–123, December 1992.

19. J. Lehoczky, L. Sha, and Y. Ding. Enhancing aperiodic responsiveness in hard real-time environment. In *Proceedings IEEE Real-Time Systems*, pages 261–270, December 1987.

20. J. Migge, Alain Jean-Marie, and N. Navet. Timing analysis of compound scheduling policies : Application to Posix1003.1b. *Accepted for publication in Journal of Scheduling*, to appear in 2001.

21. J.M. Migge and A. Jean-Marie. Timing analysis of real-time scheduling policies: a trajectory based model. Technical Report 3561, INRIA, 1998.

22. J.M. Migge and A. Jean-Marie. Non-preemption, critical sections and round robin. Technical Report 3678, INRIA, 1999.

23. N. Navet and Y.-Q. Song. Reliability improvement of the dual-priority protocol under unreliable transmission. *Control Engineering Practice*, 7(8):975–981, 1999.

24. N. Navet, Y.-Q. Song, and F. Simonot. Worst-case deadline failure probability in real-time applications distributed over CAN (Controller Area Network). *Journal of Systems Architecture*, 46(7):607–618, 2000.

25. M.-A. Peraldi and J.-D. Decotignie. Combining real-time features of local area network fip and can. Technical report, École Polytechnique de Lausanne, 1996.

26. F. Riera. *Monstre: A Microkernel orientat a les necessitats de sistemes de temps real empotrats*. Final year project, Universitat de les Illes Balears, Departament de Ciències Matemàtiques i Informàtica, 1997.

27. Society of Automotive Engineers SAE. Class c application requirement considerations. Technical report, Society of Automotive Engineers, Jun 1993. J2056/1.

28. SAE Vehicle Network for Multiplexing and Data Communications Standards Committee. Class b data communications network interface - sae j1850 standard, 1996. Rev. NOV96.

29. S.P. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1):27–60, 1989.

30. K. Tindell and H. Hansson. Babbling idiots, the dual priority protocol, and smart can controllers. In *2^{nd} international CAN Conference, ICC'95*, pages 7.22–7–28, september 1995.

31. K. Zuberi and K.G. Shin. Scheduling on controller area network for real-time cim applications. *IEEE Transactions on Robotics and Automation*, 13(2):310–314, April 1997.