HAL open science

Simulation intervals for uniprocessor real-time schedulers with preemption delay

Joel Goossens, Damien Masson

To cite this version:

Joel Goossens, Damien Masson. Simulation intervals for uniprocessor real-time schedulers with preemption delay. The 30th International Conference on Real-Time Networks and Systems (RTNS) 2022, Jun 2022, Paris, France. pp.36-45, 10.1145/3534879.3534887 . hal-03856537

HAL Id: hal-03856537

https://univ-eiffel.hal.science/hal-03856537v1

Submitted on 16 Nov 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Simulation intervals for uniprocessor real-time schedulers with preemption delay

Joël Goossens
Université libre de Bruxelles (ULB)
Brussels, Belgium
joel.goossens@ulb.be

Damien Masson
Univ Gustave Eiffel, CNRS, LIGM
F-77454, Marne-la-Vallée, France
damien.masson@esiee.fr

ABSTRACT

In the framework of embedded and time-critical systems we consider the scheduling of preemptive real-time periodic tasks upon uniprocessor. We consider the notion of *simulation interval*, a finite time interval such that the schedule starts to repeat in a cycle. The interest is to design a time interval which includes *all* possible (reachable) schedule states. Our study focuses on a model where preemption costs are *explicitly* considered, i.e., the time required by the real-time operating system (RTOS) or the hardware to load the context of execution of preempted jobs. We present and prove correct a simulation interval for asynchronous arbitrary deadline tasks *with* preemption delays and that holds for *any* deterministic and memoryless scheduler upon uniprocessor. This first contribution is valid for all schedulers one can imagine (but deterministic and memoryless), including non-preemptive schedulers, not work-conserving, not necessarily popular "real-time" ones. We then consider a particular case, regarding the preemption delays, for EDF scheduling for which we extend the work of Leung and Merril [25] showing that $[0, O^{\max} + 2 \cdot H)$ is a simulation interval (significantly shorter than the general result). We also show that $[0, S_n + H)$ from [17] remains a simulation interval for fixed task priority (FTP) schedulers. Before concluding we open a discussion on the scope of the results, paths for reducing pessimism as well as other potential results opened by this work.

CCS CONCEPTS

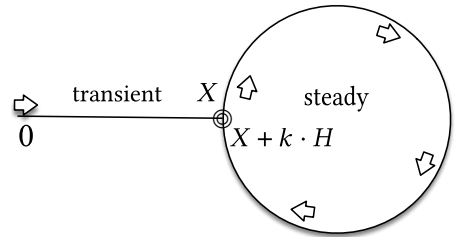
• **Computer systems organization** → **Real-time operating systems**.

KEYWORDS

Scheduling theory, uniprocessor, preemption delay

ACM Reference Format:
Joël Goossens and Damien Masson. 2022. Simulation intervals for uniprocessor real-time schedulers with preemption delay. In *Proceedings of The 30th International Conference on Real-Time Networks ans Systems (RTNS '22)*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

Figure 1: A periodic schedule of periodic tasks.



1 INTRODUCTION

In this research we study the scheduling of asynchronous arbitrary deadline preemptive hard real-time periodic task sets upon uniprocessor systems. We consider *preemptive* tasks/jobs in the sense that a higher priority task/job can preempt the current task/job in order to execute the highest priority one. The originality of this work, regarding the state-of-the-art, is the fact that the task model considers *preemption delays*, i.e., the duration required by the real-time operating system (RTOS) or the hardware to resume a task/job after a preemption.

We study the notion of *simulation interval* with the following definition:

Definition 1 (Simulation interval). *A finite and safe time interval* $[0, b)$ *such that the schedule start to repeat in a cycle.*

In this work, we consider the notion of simulation interval for *feasible* schedule. Note that infeasible schedules are not necessarily periodic, but this question is beyond the scope of this work.

A simulation interval is a window of time during which *all* reachable states of the system have been observed. A schedule consists of two parts, a first one called the *transient phase*, the time interval $[0, X)$, during which the succession of states appears only once, and a second one called the *steady phase*, the time interval $[X, X + kH)$, during which the succession of states is periodic. Most of the time[1] the period of the schedule is a multiple of the hyperperiod (see

[1] In particular cases, it is possible to define schedulers whose periodicity does not depend on the arrival of tasks, but in practice they make no sense.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
RTNS '22, June 07–08, 2022, Paris, France
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

Section 3 for a formal definition). This is illustrated in Figure 1, with X and k as unknown constants and H as the hyperperiod. We are interested in finding exact values for X and k or at least an upper-bound for $X + k \cdot H$.

The notion of simulation interval should not be confused with the notion of *feasibility interval*:

Definition 2 (feasibility interval). *A finite interval $[a, b)$ such that if all the deadlines of jobs released in the interval are met, then the system is schedulable.*

Informally speaking, the feasibility interval includes the worst-case schedule states regarding schedulability like, e.g., the first busy period for synchronous arbitrary deadlines and fixed task priority (FTP) schedulers [24] while the simulation interval includes *all* possible (reachable) schedule states.

We will also use the notion of *exact* simulation interval:

Definition 3 (Exact simulation interval). *An* exact simulation interval *is a simulation interval $[0, b)$ such that there is no simulation interval $[0, c)$ with $c < b$.*

Organization. The remainder of this paper is structured in the following manner. Section 2 presents a motivations of our work. Section 3 presents our task model and assumptions. Section 4 describes related work. Section 5 presents our contributions. Section 6 presents discussions on the scope of the results, paths for reducing pessimism as well as other potential results opened by this work. Lastly, Section 7 concludes our work.

2 MOTIVATIONS

We motivate here the two axis of our work: first why we are interested in the simulation intervals, then why we consider preemption delays in our model.

2.1 Simulation Intervals

The need to have a simulation interval is justified each time we need to study the *significant* part of a schedule and thus avoid the anecdotal (non representative) part. This could be for performance or benchmark studies. Design tools (such as Cheddar [32, 33] or Tina [6]) often require to provide the simulation interval as input. When the worst-case scenario is known (or sometimes several cases to cover the worst-case, with a brute-force approaches see e.g. [20]) the notion of simulation interval allows to have an *exact* schedulability test. For example Nasri et al. [29] use it to have an exact simulation-based schedulability test for the FIFO scheduler and tasks with offsets. Unfortunately, regarding the tasks duration, the worst-case scenario does not necessarily corresponds to consider the worst-case execution time (WCET — see Section 3), i.e., schedulers are not necessarily C-sustainable [9]. We will discuss this aspect before our conclusion (see Section 6). In *offline* approaches such as in the work of Xu and Parnas [38], it is fundamental to know a simulation interval in order to build and evaluate the *whole* schedule (i.e., until the end of the steady state). Regarding our scheduling problem, asynchronous periodic tasks with preemption delays, unfortunately the synchronous case is not the worst-case scenario. This fact motivates specifically the study of *asynchronous* scenarios.

2.2 Preemption Delays

A common assumption in scheduling theory, implicitly or not, since the seminal work of Liu and Layland [27], is that we neglect the time needed for preemption, i.e., the time needed for the system (bare metal or operating system) to switch from one task to another. More generally, all disturbances related to the hardware or the operating system, which could be called *scheduling costs*, are mostly neglected when studying the schedulability of real-time systems. Precisely, it is often implicitly assumed that these costs are an integral part of the worst-case execution time (WCET) and therefore indirectly included in the analysis [27]. However, in the case of preemption, the cumulative time spent in operating system routines or interrupt handlers depends on the number of preemptions and therefore on the scheduling. One could argue that the scheduling costs can easily be integrated in the WCET with the argument that the price is paid by the task that initiates the preemption and not by the task that suffers it (the price is the cost of the context switch to execute the new task plus the one to resume the preempted task). Then, since with fixed task priority (FTP) schedulers or even fixed job priority schedulers such as EDF, a task can only preempt one task per activation (when it is activated), so the cost is limited and can be integrated into the WCET.

A more detailed study of the mechanisms at the origin of what can be called the scheduling cost goes against this argument.

We can distinguish several components to the scheduling cost, for example:

- C_{in} the cost of adding a job to the queue,
- C_{start} the cost of starting a job,
- C_{restart} the cost of restarting a job after a preemption,
- C_{out} the cost of terminating a job.

Depending on the system, most of these components are constant or can be bounded and occur only once per job. For example, if the system is handling the arrival of tasks with interrupts and has sufficient priority levels for interrupts, C_{in} is only paid when the job effectively starts, so it does not interfere with the other jobs (this is called integrated management). C_{start} and C_{out} depend on the jobs, they correspond respectively to the loading of the execution context of the job and to its release. Depending on the implementation, they can cover different types of overhead for the system, but it is reasonable to think that they can be integrated into the WCET. C_{restart} is more problematic, as it depends on each task, and is paid after each preemption. Because of that, it is not necessarily equivalent, in terms of duration, for a task τ_i to preempt a task τ_j or another task τ_k. We so think it is important to add C_{restart} to the model and the analysis.

Another issue with these scheduling costs is that they correspond, or may correspond, to the execution of operating system routines, and therefore to non-preemptive or at least *"atomic"* sections of code, i.e., they can be interrupted, but they will then have to be re-executed in their totality.

The model considered in this paper is a first step towards a model that fully integrates the operating system (or the hardware) interference during task preemption. In order to simplify the problem we have arbitrarily chosen to consider only C_{restart}, which we note α_i for the task τ_i (see Section 3). We also chose to consider its execution completely non-preemptive, i.e., once the system starts to

reload the context of a previously preempted task τ_i, no preemption can take place for a duration equal to α_i. In future work, we may relax this constraint partially to allow preemption on the condition that we must re-execute the reloading section completely later. We will also have to consider the same constraints for the other components of the scheduling cost.

3 MODEL AND ASSUMPTIONS

In this work we consider the scheduling of asynchronous periodic task set with preemption delays upon uniprocessor. More precisely, we assume there is a delay $\alpha_i \in \{\mathbb{N} \cup 0\}$ when a job of τ_i is *resumed* after a preemption. It represents the time required to load the context of execution of the preempted job. We assume this activity non-preemptive.

More formally a task τ_i is characterised by the tuple $(O_i, C_i, T_i, D_i, \alpha_i)$. A system Sys $= \{\tau_1, \ldots, \tau_n\}$ is a task set, where every task τ_i is defined by:

- $O_i \in \{\mathbb{N} \cup 0\}$ the task offset, i.e., the release date of the first job of τ_i,
- $C_i \in \mathbb{N}$ the worst-case execution time (WCET), i.e., the maximum cumulative CPU units required for a job of τ_i to be executed,
- $T_i \in \mathbb{N}$ the task period, the kth job $\tau_{i,k}$ of τ_i is released at the instants $O_i + (k-1)T_i, k \in \mathbb{N}$,
- $D_i \in \mathbb{N}$ is the *relative* deadline and represents the timing constraint of a task: the $k^{\text{th}}, (k \in \mathbb{N})$ job $\tau_{i,k}$ of τ_i must be completely executed in the window $[O_i + (k-1)T_i, O_i + (k-1)T_i + D_i)$. We consider *arbitrary deadline* where there is no constraint between task deadline and period. Particular cases include *constrained deadlines* where $\forall i \in \{1 \ldots n\}, D_i \leq T_i$, and *implicit deadlines* where $\forall i \in \{1 \ldots n\}, D_i = T_i$. When considering arbitrary deadline it is admitted to have $D_i > T_i$, we assume to schedule the various active jobs of the same task in the FIFO order (the oldest job first).
- $\alpha_i \in \{\mathbb{N} \cup 0\}$ represents the time required by the RTOS/HW to load *non-preemptively* the context of execution of each preempted job of task τ_i.

The following parameters can be deduced[2]:

- $H \doteq \text{lcm}(T_1, \ldots, T_n)$ is the hyperperiod of the system, where lcm denotes the least common multiple,
- $H_i \doteq \text{lcm}\{T_j \mid j = 1, \ldots, i\}$,
- $O^{\max} \doteq \max_{i=1,\ldots,n}(O_i)$ is the largest offset,
- $\alpha_{\max} \doteq \max_{i=1,\ldots,n}(\alpha_i)$ is the largest preemption delays,
- $U \doteq \sum_{i=1}^{n} C_i/T_i$ is the processor utilization factor.

We will use the notations:

- $(a)_0$ meaning $(a)_0 \doteq \max(a, 0)$,
- $[x, y)$ denoting an half-open interval: $\{z \mid x \leq z < y\}$

Please notice that, in our model, we consider *discrete* time.

4 RELATED WORK

We will start by reviewing the works on simulation intervals, and then we will look at the works considering the overheads related to the scheduler and in particular to preemptions.

[2]where $\doteq$ means "equals by definition".

4.1 Simulation intervals

In the uniprocessor context, assuming an utilisation factor less than or equal to 1 and for systems of independent tasks with constrained deadlines and scheduled with EDF, the pioneering work [25] shows that the interval $[0, O^{\max} + 2H)$ is a simulation interval. Moreover, the *transient phase* is included in the window $[0, O^{\max} + H)$ and the *steady phase* is contained in the scheduling produced in the interval $[O^{\max} + H, O^{\max} + 2H)$.

For fixed task priority (FTP) schedulers, it is shown in [26] that the same interval applies. The result is extended to arbitrary deadlines in [18]. A better bound is presented in [17] and is $[0, S_n + H)$ where S_n is given by the recursive equation $S_1 \doteq O_1, S_i \doteq \max(O_i, O_i + \lceil \frac{(S_{i-1} - O_i)_0}{T_i} \rceil T_i)$.

A more general result, which holds for resource-sharing tasks with precedence constraints, constrained deadlines, and assumes only a work-conserving scheduler, was first shown in [13] then extended to parallel tasks in [3]. In that work, it is shown that an *exact* simulation interval (see Definition 3) is $[0, \theta_c + H)$ where θ_c is the date of the last idle time of the *transient phase*. Moreover, it is shown that θ_c belongs to the interval $[0, O^{\max} + H)$. It follows that the *steady phase* corresponds to the first time window of length H containing exactly $H(1-U)$ idle times in the interval $[0, O^{\max} + 2H)$.

Several works have also addressed the problem for uniform multiprocessor platforms with global scheduling. In [14], it is shown that for constrained deadlines, in the synchronous case the simulation interval is bounded by $[0, H)$, while in the asynchronous case the interval $[0, S_n + H)$ given in [17] still held. In [15], the authors modify the definition of S_n to give a simulation interval for arbitrary deadlines. The bound becomes $[0, \hat{S}_n + H)$ where $\hat{S}_n$ is given by the recursive equation $\hat{S}_1 \doteq O_1, \hat{S}_i \doteq \max(O_i, O_i + \lceil \frac{(\hat{S}_{i-1} - O_i)_0}{T_i} \rceil T_i) + H_i$. The result is extended to heterogeneous (unrelated) multiprocessor systems in [16]. A result is given in the case of offline schedulers for systems with precedence constraints in [4]. It is generalized to EDF and to independent tasks in [30]: $[0, O^{\max} + H \prod_{i=1}^{n}(C_i + 1))$.

Finally, a very general result for any deterministic and memoryless scheduler is given in [19] and is $[0, H \prod_{i=1}^{n}((O_i + D_i - T_i)_0 + 1))$. This interval works for task systems with multiple dependencies, defined as *structural constraints* (mutual exclusions, precedence constraints, self-suspension, non-preemptive tasks, etc.).

In a recent paper [22], the authors propose a method to reduce the size of the simulation interval. They provide an *exact simulation interval*. However, the computation of this interval is heavily time consuming, involving a factorial time complexity computation of a set of points defined by an exponential number of linear constraints. The authors show that, despite its intractable time consuming computation, the size of the exact interval grows much slower than the one from [19] with the number of tasks.

4.2 Taking into account the cost of preemptions

Many works have proposed ways to integrate scheduling delays or even other overheads related to the hardware platform or the operating system. Different approaches exist. Some of them consist in bounding this interference in order to add it to the schedulability analysis as part as the WCET or as a blocking term, inputting cost overheads either to the preempted tasks, the preempting ones or

both. Some other approaches propose different task models with associated analyses. Finally some works adapt the scheduler algorithm to make it aware of the existence of such delays.

It is interesting to note that a part of the scheduling delays are external to the chosen scheduling policy, i.e., the overheads are imposed by the hardware platform or the operating system. They must nevertheless be taken into consideration in the schedulability analysis or more generally in the scheduler performance evaluation. As we point it out in this paper, they have an impact on the simulation interval and the periodicity of the schedule.

The first works to have been interested in the impact of the scheduler implementation and the kernel on scheduling are those of [21]. In that paper, the authors describe four distinct generic methodologies for implementing a scheduler: *Integrated Interrupt Event-Driven Scheduling*, *Nonintegrated Interrupt Event-Driven Scheduling*, *Timer-Driven Scheduling* and *Timer-Driven Scheduling with counter*. The first one assumes that tasks are activated by external hardware interrupts. The second one is a special case where there is no prioritised interrupts on the platform. The third one corresponds to what is also referred to in the literature as *tick scheduling* where the scheduler is woken up periodically every quantum of time and processes the events that have occurred since its last wake-up. In addition to the overhead experienced at each tick, due to the granularity of the triggering timer, transient priority inversions can also occur. The last one is a refinement of the previous one where instead of a periodic timer, an updated counter is used allowing a wake-up only when scheduling events occur (next activation, deadline, etc.). For each of them, they detail the cost components inherent to the scheduler, and extend the sufficient popular schedulability condition for FTP and constrained deadlines ($\forall_i, \exists t \leq D_i$ s.t. $\sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + B_i \leq t$) to take them into account either in the C_j term or in the B_i one. The extended conditions are first given under the assumption of a preemptive kernel, then a non-preemptive kernel, which adds an extra component to the blocking term B_i.

The works in [11] point out the pessimism of the sufficient conditions developed in the preceding works and gives a finer analysis of the overhead of a scheduler implemented according to the tick scheduling paradigm.

Other works (see, e.g., [5, 8, 37]) have considered the problem of limiting the number of preemptions or the one of computing a bound on the number of possible preemptions, permitting an integration of a maximum blocking time linked to the system interference in the schedulability analysis.

Another source of disturbance related to preemptions analysed in the literature is related to the use of caches. While caches permit a reduction of response times, they also introduce a significant source of variability in WCETs due to the possibility for the task to suffer additional delays due to cache misses. The runtime overhead associated with cache misses caused by an early preemption is called Cache Related Preemption Delay (CRPD)[12]. The strategy for taking CRPDs into account is to bound them for inclusion in the feasibility analysis of FTP schedulers. To calculate a bound, one can use Evicting Cache Blocks (ECBs, the blocks used by the preempting task and which can replace those of a preempted task [12]) or Useful Cache Blocks (UCBs, the blocks used by a preempted task which are reused later, and therefore potentially deleted during a preemption [23]). Some works combine the two approaches [1, 34, 35]. The case of EDF is studied in [28] by extending the feasibility test based on the Demand Bound Function (DBF). It is shown in [31] that optimal scheduling with CRPDs taken into account is an NP-hard problem. More recently, a feasibility interval is given for this problem in [36].

What distinguishes our study from these works is that we consider the overhead of preemption not as a special case where the WCET is increased, but by adding a non-preemptive code block after each preemption.

In his PhD thesis and in [39], Meumeu proposes an approach to include the cost of reloading a task after a preemption in the feasibility analysis of a system scheduled with FTP. In his model, this reloading cost is different for each task, and consists in an *"atomic"* sequence of operations (it can be preempted but then all the sequence must be re-executed later). He shows that with this model, the critical instant cannot be characterised, that the worst-case response time of a task can occur in the transient phase, and that the Optimal Priorities Assignment algorithm (OPA) compatibility conditions [2] are not respected. He proposes an optimal priority assignment algorithm.

In his PhD Thesis, Bimbard [7] considers the kernel overheads of an OSEK kernel (fixed priority scheduling), taking into account activation, termination and preemption delays. For a given task, activation and termination are extra duration added to the WCET of a task while preemption delays are taken into account at each activation of higher priority task. A worst case response time analysis is proposed with a bound on the number of preemptions that makes the synchronous scenario still the worst case scenario at the price of some pessimism.

This Research. In this paper *(i)* we model preemption delays as task-dependant non preemptive additional code sections executed for a task when it is resumed after a preemption, *(ii)* we present and prove correct a simulation interval for asynchronous arbitrary deadline tasks *with* such preemption delays and that holds for any deterministic and memoryless scheduler upon uniprocessor, *(iii)* for particular case, regarding the preemption delays, and EDF scheduling we extend the work of Leung and Merril [25] showing that $[0, O^{\max} + 2 \cdot H)$ is a simulation interval (significantly shorter than the general result). Lastly, *(iv)* for FTP schedulers we also extend the work [17] and we show show that $[0, S_n + H)$ remains a simulation interval.

5 CONTRIBUTIONS

A first contribution of this paper is to provide a *simulation interval* (see Definition 1) for the scheduling of periodic tasks with preemption delays. We build upon the work of [19] where the authors provided a simulation interval and more interestingly a technique to do so.

The former work concerned multiprocessor platforms and dependent tasks. Here the novelty is to consider preemption delays for uniprocessor platform. In Section 5.1 we will introduce the technique of [19], in Section 5.2 we present and prove correct a simulation interval for tasks with preemption delays and that holds for any deterministic and memoryless scheduler (see Definition 5).

5.1 Summarising the technique of [19]

The authors consider multiprocessor scheduling of preemptive periodic real-time tasks *without* preemption delays. They first formalise the notions of *system state, system pre-state* and *deterministic and memoryless scheduler* with Definitions 4 and 5.

Definition 4 (State and pre-state of a system [19]). *The state of a system of n tasks can be defined as a $(2n)$-tuple*

$$S \doteq \left\langle C_{\text{rem}_1}, \ldots, C_{\text{rem}_n}, \Omega_1, \ldots, \Omega_n \right\rangle,$$

where

- *Ω_i is the local clock of τ_i, undefined before O_i, initialized at 0 at the time O_i, being reset at every new request of the task. Formally, at time $t \geq O_i$, $\Omega_i \doteq (t - O_i) \mod T_i$,*
- *while C_{rem_i} is the cumulative remaining work to process for τ_i (can be greater than C_i since it is allowed to have $D_i > T_i$).*

We will see that we need to consider arbitrary *deadlines even if the original system under study have constrained deadlines (see Lemma 7 for details) consequently we assume to schedule the various jobs of the same task in the* FIFO *order (the oldest job first).*

The pre-state of a system of n tasks can be defined as a $(2n)$-tuple $\hat{S} \doteq \left\langle \hat{C}_{\text{rem}_1}, \ldots, \hat{C}_{\text{rem}_n}, \Omega_1, \ldots, \Omega_n \right\rangle$, where:

- *Ω_i is the same local clock as in the state S of the system,*
- *$\hat{C}_{\text{rem}_i}$ is the remaining work to process for τ_i <u>not</u> taking the releases at the considered instant into account.*

We can formalize the remaining work in state and pre-state, for any $t \geq O_i$ as follows:

$$\begin{aligned}
\hat{C}_{\text{rem}_i}(t) &\doteq 0, \forall t \leq O_i \\
C_{\text{rem}_i}(t) &\doteq \hat{C}_{\text{rem}_i}(t) + C_i \text{ if } \Omega_i = 0 \\
& \hat{C}_{\text{rem}_i}(t) \text{ otherwise} \\
\hat{C}_{\text{rem}_i}(t+1) &\doteq C_{\text{rem}_i}(t) - 1 \text{ if } \tau_i \text{ executed on } [t, t+1) \\
& C_{\text{rem}_i}(t) \text{ otherwise}
\end{aligned}$$

Definition 5 (Deterministic and memoryless scheduler [19]). *A scheduler is* deterministic and memoryless *if, and only if, the scheduling decision*[3] *at time t is* unique *and* univocally *defined by the state of the system at time t.*

Consequently, the state of the system must contain *all* information required by the scheduler to take its decision but *only* the strictly required and *minimum information* in order to bound as much as possible the number of distinct states.

Then the authors show that considering only the *synchronous* scenario of a *transformed and arbitrary deadline* task set is sufficient to enumerate all feasible schedules of a system, thanks to Lemma 7 (and Definition 6).

Definition 6 (Set of feasible schedules [19]). *We define the function $\mathcal{F}$ such that $\mathcal{F}(S)$ is the set of all feasible schedules obtained by deterministic and memory-less schedulers for task system S.*

Lemma 7 (Lemma 1 in [19]). *Let S be a set of independent tasks with $\forall i \in 1, \ldots, n, O_i \geq 0$. We denote O_i the offset of the task τ_i and D_i its relative deadline. Let S' be the same system, except for the release dates given by $O'_i = 0$ and the relative deadlines $D'_i = D_i + O_i$. The set of feasible schedules of S is included in the set of feasible schedules of S', i.e., $\mathcal{F}(S) \subseteq \mathcal{F}(S')$.*

Please notice that, while the original system S can be composed of *constrained* deadlines, the synchronous system S' could be composed of *arbitrary* deadlines.

Based on the notions of state and pre-state of a system, the authors show, in Lemma 8, that if two states or pre-states are identical, then the scheduling decision of any deterministic and memoryless schedulers are identical, consequently the schedule repeats. This allows the authors to bound the number of distinct (pre-)states in Lemma 9.

Lemma 8 (Lemma 2 in [19]). *For synchronous task systems, if two pre-states are identical, then the scheduling decision of a deterministic and memoryless scheduler is the same.*

Lemma 9 (Lemma 3 in [19]). *Any feasible schedule of a* synchronous *arbitrary deadline independent task system generated by a deterministic and memoryless scheduler reaches a cycle*[4] *at or prior to*

$$H \cdot \prod_{i=1}^{n} ((D_i - T_i)_0 + 1).$$

Lastly, the authors combine Lemma 7 and Lemma 9 to provide and prove correct a simulation interval for *asynchronous* systems with the Theorem 10.

Theorem 10 (Theorem 1 in [19]). *Any feasible schedule of an* asynchronous *independent tasks system generated by a deterministic and memoryless scheduler reaches a cycle*[4] *at or prior to*

$$H \cdot \prod_{i=1}^{n} ((O_i + D_i - T_i)_0 + 1).$$

Recap. For the determination of simulation intervals, the technique of [19], is based on the formalization of the notion of system state in a mininalist way which combined with a deterministic and memoryless scheduler leads to enumerate and bound the distinct possible (reachable) states.

5.2 A first contribution: a simulation interval for deterministic schedulers with preemption delay

In this section we extend the work of Goossens et al. in [19] in the sense that for each task τ_i we have an additional α_i parameter the reload delay.

Following the same arguments as Goossens et al. in [19] we design and proof correct the following simulation interval:

Theorem 11 (Simulation interval). *Any feasible schedule of asynchronous arbitrary deadline periodic tasks with preemption delays generated by a deterministic and memoryless scheduler upon uniprocessor platform with a reload delay of α_i for the task τ_i reaches a cycle*[4] *at or prior to:*

$$H \cdot (n+1) \cdot (\alpha_{\max} + 1) \prod_{i=1}^{n} ((O_i + D_i - T_i)_0 + 1). \tag{1}$$

[3] By scheduling decision at time t we mean what the scheduler/system decides to do during the slot $[t, t+1)$: idle the processor, execute a task, reload a preempted task.

[4] i.e., all reachable states are visited.

In order to prove that property we need first to extend Definition 4, the system (pre-)state to consider preemption delay. We have to design it as minimal as possible to limit as much as possible the number of distinct (pre-)states. Our proposition is to consider two additional parameters:

- $r \in \{0, 1, \ldots, n\}$ the task index of the running/reloaded task (if any) or 0 if the processor is idle. As we have non-preemptive reloading phases, the decision at time $t+1$ depends on the decision taken at time t.
- rem-reload $\in \{0, 1, \ldots, \alpha_{\max}\}$ the remaining reload time (0 means that a task executes).

More formally:

Definition 12 (State and pre-state of a system). *The state of a system of n tasks can be defined as a $(2n)$-tuple*

$$S \doteq \left\langle C_{\text{rem}_1}, \ldots, C_{\text{rem}_n}, \Omega_1, \ldots, \Omega_n \right\rangle,$$

$r \in \{0, 1, \ldots, n\}$, *and* rem-reload $\in \{0, 1, \ldots, \alpha_{\max}\}$. *See Definition 4 for details (concerning the already defined attributes).*

We will now formalize the remaining reload time in state, as follows:

rem-reload$(t+1) \doteq$

$$\text{rem-reload}(t) - 1 \qquad \textit{if } \text{rem-reload}(t) > 0 \tag{2}$$

$$\alpha_\ell \qquad \textit{if } r(t+1) = \ell > 0 \wedge r(t) \neq \ell \wedge (C_{\text{rem}_\ell}(t+1) \% C_\ell) \neq 0 \tag{3}$$

$$0 \qquad \textit{otherwise} \tag{4}$$

Equation 3 corresponds to the begin of an reload region such that τ_ℓ was preempted in the past. Indeed we consider arbitrary deadlines consequently several jobs of the same task can be active simultaneously, $C_{\text{rem}_\ell}(t+1)$ represents the cumulative remaining computation time. Notice that if $C_{\text{rem}_\ell}(t+1)$ is not a multiple of C_ℓ a job of τ_ℓ is resumed otherwise this is a new job of τ_ℓ since at time t a different task ($r(t) \neq \ell$) was executed/reloaded[5].

Lemma 13. *Any feasible schedule of a synchronous arbitrary deadline task system generated by a deterministic and memoryless scheduler with preemption delay reaches a cycle[4] at or prior to*

$$H(n+1)(\alpha_{\max}+1) \prod_{i=1}^{n} \left((D_i - T_i)_0 + 1\right).$$

Proof. For parameters in common between Definition 4 and Definition 12 (C_{rem_i}'s and Ω_i's) from [19] we know that, regarding the states and pre-states, that in the worst case after $\prod_{i=1}^{n}(D_i - T_i)_0 + 1$ hyperperiods the state or the pre-state repeats. We must complete the arguments considering the two extra parameters (r and rem-reload): we need that value r repeats as well as the value rem-reload. Since r is limited to $n+1$ values and rem-reload is limited to $\alpha_{\max}+1$ values. We conclude that after a maximum of $(n+1) \cdot (\alpha_{\max}+1) \prod_{i=1}^{n} \left((D_i - T_i)_0 + 1\right)$ hyperperiods the schedule (if feasible) repeats. □

In order to have similar result for *asynchronous* systems we need to extend Lemma 7 for systems with reload delay.

[5] if the scheduler has decided to idle the processor at time t (i.e., $r(t) = 0$) we decide arbitrarily to make a reload in this case.

Lemma 14. *Let S be a set of independent tasks with $\forall i \in 1, \ldots, n$, $O_i \geq 0$. We denote O_i the offset of the task τ_i and D_i its relative deadline. Let S' be the same system, except for the release dates given by $O'_i = 0$ and the relative deadlines $D'_i = D_i + O_i$. The set of feasible schedules of S is included in the set of feasible schedules of S'.*

Proof. A feasible schedule for S is also a feasible schedule for the system S', since the absolute deadlines are identical, we just release jobs earlier which doesn't affect the validity of the solution. □

Theorem 11 follows from Lemma 14 and Lemma 13.

5.3 Shorter interval for EDF and FTP schedulers in a particular case

Introduction. In this section we will study the particular case of *constrained* deadlines and binary delays ($0 \leq \alpha_i \leq 1$, see Definition 17). We will see that it is not straightforward to define shorter intervals — in comparison with $[0, H \cdot (n+1) \cdot (\alpha_{\max}+1) \prod_{i=1}^{n}((O_i + D_i - T_i)_0 + 1))$ — with preemption delays. We will first consider the case of EDF and see why the Leung and Merril [25] interval, $[0, O^{\max} + 2H)$, for EDF scheduling is not a simulation (or feasibility) interval. We will exhibit the phenomenon which occurs because of preemption delays and which invalidates the Leung and Merril's properties. Lastly, we consider the case of FTP schedulers.

Let's first recap the work of Leung and Merril. In [25], the authors consider the EDF scheduling of periodic *constrained* deadline task sets upon uniprocessor. They design and prove correct the $[0, O^{\max} + 2H)$ simulation (and feasibility) interval. They first prove Lemma 1 required to prove Lemma 2 the main property:

Lemma 15 (Lemma 1 in [25]). *Let S be schedule of a task system R constructed by the EDF algorithm. Then for each tasks τ_i and for each time instant $t_1 \geq O_i$, we have $e_{i,t_1} \geq e_{i,t_2}$, where $t_2 \doteq t_1 + H$, and $e_{i,t}$ is the amount of time for which τ_i has executed since its last request.*

Lemma 16 (Lemma 2 in [25]). *Let S be schedule of a task system R constructed by the EDF algorithm. If R is feasible, then $C_S(R, t_1) = C_S(R, t_2)$, where $t_1 = O^{\max} + H$, $t_2 = O^{\max} + 2H$ and $C_S(R, t)$, to be the n-tuple $(e_{1,t}, \ldots, e_{n,t})$.*

In the following, we will show that, unfortunately, Lemma 15 and consequently Lemma 16 are not satisfied with preemption delays.

Consider, for instance, the system described by Table 1. Figure 2 corresponds to the EDF schedule, in this case the periodicity starts immediately (at time origin) but the period is $2 \cdot H = 24$. In Figure 2, we represent a task request by a ↓ (down arrow); we represent the task deadline with a ◯ (circle); white rectangles represent task execution, grey rectangles represent reload activities (⋈ represents priority inversion a notion that will be introduced later).

This example contradicts Leung and Merril's property (Lemma 15) required to prove that schedule repeats from $O^{\max} + H$ with a period H, since we have: $0 = e_{4,7} < e_{4,19} = 1$.

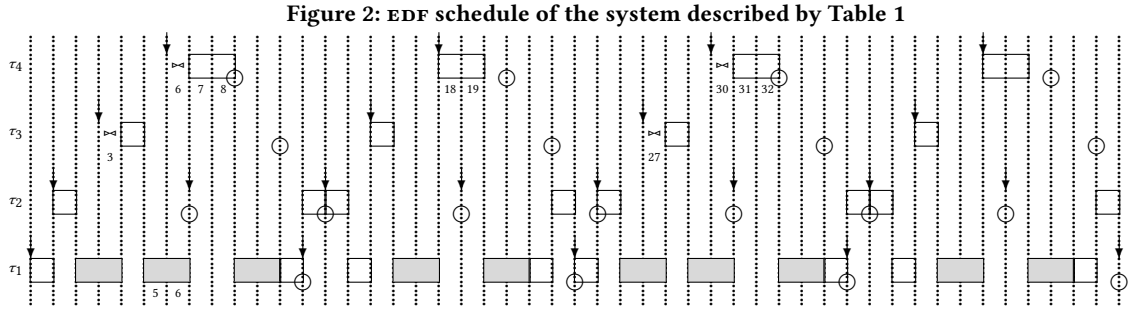
Informally speaking, the progress of task τ_4 at time instant 7 is strictly smaller than the progress of task τ_4 an hyperperiod later (at time instant 19).

Consequently the system state at time instant $O^{\max} + H = 18$ is not the same an hyperperiod later (at time instant 30), in particular at time instant 30 task τ_1 must complete its non-preemptive reload

Table 1: Task set characteristics

	T_i	D_i	C_i	O_i	α_i
τ_1	12	12	2	0	2
τ_2	6	6	1	1	2
τ_3	12	8	1	3	2
τ_4	12	3	2	6	2

Figure 2: EDF schedule of the system described by Table 1



activity while at time 18 task τ_1 can be preempted (by task τ_4) *immediately*. Of course we could extend the Leung and Merril's formal definition of system state (C_S) to take account of reloads as we did in the Definition 12 to formally have $C_S(R, t_1) \neq C_S(R, t_2)$ but this is not required.

Priority inversion. We will now exhibit more precisely the phenomenon which is problematic: the *priority inversion*. On Figure 2, $\bowtie$ represents priority inversion at time instants $3, 6, 3 + 2H = 27, 6 + 2H = 30, 3 + 4H, 6 + 4H, \ldots$ where an higher priority job is delayed by a lower priority one.

Binary delays. An interesting particular case is to consider *binary delays* because priority inversion is impossible (we will prove that in this section), and for which shorter intervals can be identified, which is a second contribution of this paper.

Definition 17 (Binary delays). *In this model the preemption delay are equal to zero or one:* $\forall i\ \alpha_i \in \{0, 1\}$.

Proposition 1 (No priority inversion with binary delays). *For binary delays and discrete time systems priority inversion is impossible, i.e., a lower priority job/task cannot interfere in the execution of higher priority job/task.*

Proof. If we resume a task/job during the time-slot $[t, t + 1)$ we know there is no higher priority task/job at time t, moreover the resumption will be completed at time $t + 1$ in order to schedule immediately an higher priority task/job (if any). □

For this particular case we can generalise the properties of Leung and Merril. To do this we will extend the definition of $C_S(R, t)$ (system state). Please notice that we are not interested here in enumerating all distinct states but we need to compare states distant by one hyper-period.

Definition 18 ($\tilde{C}_S(R, t)$). *Let S be a schedule of the task system R. The state of a system of n tasks can be defined as*

$$\tilde{C}_S(R, t) \doteq \langle e_{1,t}, \ldots, e_{n,t} \rangle, r, \textit{and}\ \text{rem-reload}$$

where at time instant t:

- *$e_{i,t}$ is the amount of time for which τ_i* has executed *since its last request,*
- *$r \in \{0, 1, \ldots, n\}$ is the task index of the running/reloaded task (if any) or* 0 *is the processor is idle,*
- rem-reload $\in \{0, 1\}$ *is the remaining reload time of τ_r if $r \neq 0$ and* rem-reload > 0. *If $r = 0$* rem-reload *is undefined.*

We have now the material to extend the Leung and Merril lemmata, However, the proofs are much more sophisticated and complex.

Lemma 19. *Let S be schedule of a task system R constructed by the* EDF *algorithm* with binary delays. *Then for each tasks τ_i and for each time instant $t_1 \geq O_i$, we have $e_{i,t_1} \geq e_{i,t_2}$, where $t_2 \doteq t_1 + H$, and $e_{i,t}$ is the amount of time for which τ_i has executed since its last request.*

Proof. We will show the property by contradiction. We assume, for the purpose of the contraction, that $\exists\ \ell_1$ s.t. $e_{\ell_1,t_1} < e_{\ell_1,t_2}$. Then there must be some time instant $t_1' < t_1$ such that task τ_{ℓ_1} is active at both instants t_1' and $t_2' = t_1' + H$, and τ_{ℓ_1} is executing at time instant t_2' but not at time instant t_1'. We will consider all cases where this can happen with binary delays. For each case (*(a)–(d)*) we will show that either the case is not possible or that it produces an *infinite* progression of requests that precede t_1' which is not possible by definition of task offsets and time origin. But first we introduce an additional notation: $S(t)$ denotes the schedule decision at time t, $S(t) = E_i$ meaning the execution of τ_i, $S(t) = R_i$ meaning the reloading of τ_i, and $S(t) = I$ meaning that the processor is idle.

Case a. $S(t_1') = E_{\ell_2}$, $\ell_2 \neq \ell_1$ (and $S(t_2' = E_{\ell_1})$), in this case task τ_{ℓ_2}, an EDF higher priority task, is active at time instant t_1' but not at time instant t_2'. But this means that $e_{\ell_2,t_1'} < e_{\ell_2,t_2'}$, and we may repeat the above argument to produce an infinite progression of task computations $\tau_{\ell_3}, \tau_{\ell_4}, \ldots$, for which no lower bound will exist for the time at which the task computations in the sequence are

active. But this is impossible since every task τ_i in the system has an initial request time O_i.

Case b. $S(t'_1) = I$ (and $S(t'_2 = E_{\ell_1})$), this is impossible since τ_{ℓ_1} is active at time instant t'_1 and EDF is work-conserving.

Case c. $S(t'_1) = R_{\ell_2}, \ell_2 \neq \ell_1$ (and $S(t'_2 = E_{\ell_1})$), task τ_{ℓ_2} is an EDF higher priority task, active at time instant t'_1 but not at time instant t'_2. Using the very same argument than case *a* we can produce an infinite progression of task computations which leads to a contradiction.

Case d. $S(t'_1) = R_{\ell_1}$ (and $S(t'_2 = E_{\ell_1})$). We will consider all subcases where this can happen with binary delays.

Case d.1. $S(t'_1 - 1) = I$ this impossible since at time $t'_1 - 1$ there is at least one active task (τ_{ℓ_1}).

Case d.2. $S(t'_1 - 1) = R_j$, $j \neq \ell_1$, The sequence $S(t'_1 - 1) = R_j, S(t'_1) = R_{\ell_1}$ is impossible, if there is no new request at time instant t'_1 we must have $S(t'_1) = E_j$ (which is not the case), if there is an request of task τ_j at time instant t_1 but then we must have $S(t'_1) = E_j$ because it is a new job that begins its execution (which is not the case either).

Case d.3. $S(t'_1 - 1) = R_{\ell_1}$ this impossible for *binary* delays.

Case d.4. $S(t'_1 - 1) = E_{\ell_2}$, $\ell_2 \neq \ell_1$. First notice that τ_{ℓ_2} cannot be active at time $t'_2 - 1$ since $S(t'_2) = E_{\ell_1}$; otherwise we must have $S(t'_2) = R_{\ell_2}$ or $S(t'_2) = E_{\ell_2}$ but we have $S(t'_2) = R_{\ell_1}$! Consequently, task τ_{ℓ_2}, an EDF higher priority task, is active at time instant $t'_1 - 1$ but not at time instant $t'_2 - 1$. But this means that $e_{\ell_2,t'_1-1} < e_{\ell_2,t'_2-1}$. Once again, using the very same argument than case *a* we can produce an infinite progression of task computations which leads to a contradiction. □

Lemma 20. *Let S be schedule of a task system R constructed by the EDF algorithm* with binary delays. *If R is EDF-schedulable, then $\tilde{C}_S(R, t_1) = \tilde{C}_S(R, t_2)$, where $t_1 = O^{\max} + H$, $t_2 = O^{\max} + 2H$.*

Proof. When R is EDF-schedulable we will first show (case *a*) that $\forall i, e_{i,t_1} = e_{i,t_2}$, and then we will show that (case *b*) if $\forall i, e_{i,t_1} = e_{i,t_2}$ then that the system states at time instant t_1 and t_2 cannot differ on the other two parameters (r and rem-load).

Case a. We will show the property by contradiction. We assume, for the purpose of the contraction, that $\exists\ \ell$ s.t. $e_{\ell,t_1} \neq e_{\ell,t_2}$. By Lemma 19, we must have $e_{\ell,t_1} > e_{\ell,t_2}$. We first show that during the time interval $[t_1, t_2)$ there is *no* idle unit. We will prove this by contradiction as well. For the purpose of the contradiction we will assume an idle unit during the time-slot $[t_1 + \Delta, t_1 + \Delta + 1)$. This imply that all task computations requested prior to $t_1 + \Delta$ have finished their execution. Consequently, $\forall i, e_{i,t_1+\Delta} = C_i$. By definition we have $\forall i, e_{i,O^{\max}+\Delta} \leq C_i$, by Lemma 19 we have $\forall i, e_{i,O^{\max}+\Delta} \geq e_{i,t_1+\Delta}$. We just proved that $\forall i, e_{i,t_1+\Delta} = C_i$. Consequently, $\forall i, e_{i,O^{\max}+\Delta} = C_i$. In other words, all task computations requested prior to $O^{\max} + \Delta$ have finished their execution, as well. Since the task requests in the interval $[O^{\max} + \Delta, t_1 + \Delta)$ and $[t_1 + \Delta, t_2 + \Delta)$ are the same and since $\forall i, e_{i,O^{\max}+\Delta} = e_{i,t_1+\Delta} = C_i$, the schedule in the interval $[O^{\max} + \Delta, t_1 + \Delta)$ and $[t_1 + \Delta, t_2 + \Delta)$ must be identical. Since $t_1 \in [O^{\max} + \Delta, t_1 + \Delta)$, $t_2 \in [t_1 + \Delta, t_2 + \Delta)$, and $t_2 - t_1 = H$; we conclude that $\forall i, e_{i,t_1} = e_{i,t_2}$ contradicting our assumption that $\exists\ \ell$ s.t. $e_{\ell,t_1} > e_{\ell,t_2}$. We have just proved that there is no idle unit in the time interval $[t_1, t_2)$. Let us remember that we assume, for the purpose of the contradiction, that $\exists\ \ell$ s.t. $e_{\ell,t_1} > e_{\ell,t_2}$. During a hyperperiod $[t_1, t_2)$ the processor is fully busy (executing or reloading tasks) but at time t_2 the task τ_ℓ is delayed in comparison with time instant t_1. As EDF decisions do not rely on the remaining computation times EDF will make the same decisions in the following hyperperiods which will sooner or later lead to a deadline miss. Hence, R cannot be EDF-schedulable on one processor. In other words, during each hyper-period what the system is able to schedule is less than the processor demand that arrives in each hyper-period. This does not mean that $U > 1$ because the processor spends time on execution *and* reloads but a backlog will accumulate and sooner or later a deadline miss will occur.

Case b. We now know that $\forall i, e_{i,t_1} = e_{i,t_2}$ we will show that the system states at time instant t_1 and t_2 cannot differ on the other two parameters (r and rem-load). Firstly, notice that if there is no active task at time t_1 (or equivalently t_2) mandatory EDF idles the processor. Secondly, since $\forall i, e_{i,t_1} = e_{i,t_2}$ the active tasks are identical at both time instants t_1 and t_2, EDF will choose the same task for execution/reloading (let's say task τ_ℓ). If the states differ (and consequently the EDF decisions) we have to distinguish between two cases:

- case *b.1*) $S(t_1) = E_\ell \neq S(t_2) = R_\ell$, and
- case *b.2*) $S(t_1) = R_\ell \neq S(t_2) = E_\ell$,

where $S(t)$ denotes the schedule decision at time t (E_ℓ meaning the execution of τ_ℓ, and R_ℓ meaning the reloading of τ_ℓ).

Case b.1. Since $S(t_1) = E_\ell \neq S(t_2) = R_\ell$ consequently $S(t_1 - 1) = E_\ell$, and $S(t_2 - 1) \neq E_\ell$. Since $\forall i, e_{i,t_1} = e_{i,t_2}$, we must have $e_{\ell,t_1-1} < e_{\ell,t_2-1}$ which contradicts Lemma 19 and proves the case *b.1* impossible.

Case b.2. If $S(t_1) = R_\ell \neq S(t_2) = E_\ell$ then $e_{\ell,t_1+1} < e_{\ell,t_2+1}$ which contradicts Lemma 19 and prove the property. □

Theorem 21 (Simulation interval for EDF). *Any EDF-schedulable asynchronous constrained deadline periodic tasks with binary delays upon uniprocessor platform reaches a cycle*[4] *at or prior to:*

$$O^{\max} + 2 \cdot H. \tag{5}$$

Proof. This is a direct consequence of Lemma 20: if the system (R) is EDF-schedulable on a single processor then $\tilde{C}_S(R, O^{\max} + H) = \tilde{C}_S(R, O^{\max} + 2 \cdot H)$ and consequently the schedule repeats from $O^{\max} + 2 \cdot H$. □

Please notice that, as the original Leung and Merril's result, it is possible to miss a (first) deadline after time instant $O^{\max} + 2 \cdot H$. To ensure the periodicity/schedulability it is necessary to test the condition: $\tilde{C}_S(R, O^{\max} + H) = \tilde{C}_S(R, O^{\max} + 2 \cdot H)$. Unfortunately, the condition $U \leq 1$ is not sufficient for system with preemption delays.

Binary delays and FTP schedulers. For fixed task priority (FTP) schedulers such as Rate Monotonic, Deadline Monotonic or OPA a much shorter simulation interval than $[0, H \cdot (n+1) \cdot (\alpha_{\max} + 1) \prod_{i=1}^{n}((O_i + D_i - T_i)_0 + 1))$ and even than $[0, O^{\max} + 2 \cdot H)$ could be designed based on the work [17] for binary delays.

The benefit of the FTP family is that we can often and easily prove properties by induction on the task set cardinality, but first we recall a definition:

Definition 22 (S_n[17]). $S_1 \doteq O_1$, $S_i \doteq O_i + \lceil \frac{(S_{i-1} - O_i)_0}{T_i} \rceil T_i$. *Informally speaking $S_i (i > 1)$ corresponds to the first release of τ_i not before S_{i-1}.*

Theorem 23 (Simulation interval for FTP). *Any FTP-schedulable asynchronous* constrained *deadline periodic tasks with binary delays upon uniprocessor platform reaches a cycle*[4] *at or prior to:*

$$S_n + H. \tag{6}$$

Proof. By induction. We first assume that, wlog, the FTP scheduler corresponds to $\tau_1 > \tau_2 > \cdots > \tau_n$. Let $\mathcal{P}(i)$ be the proposition: if the task set $\{\tau_1, \tau_2, \ldots, \tau_i\}$ is FTP-schedulable then the schedule is periodic from S_i with a period of H_i.

Base case: $\mathcal{P}(1)$ is true: if feasible the schedule for τ_1 is periodic of period $T_1 = H_1$ from the first release of τ_1 ($S_1 = O_1$).

Inductive step: For $i > 1, \mathcal{P}(i-1) \Rightarrow \mathcal{P}(i)$ is true. Assume $\mathcal{P}(i-1)$ is true for purpose of induction. We need to show that $\mathcal{P}(i)$ is true. $\mathcal{P}(i-1)$ means that if the task set $\{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ is FTP-schedulable then the schedule of $\{\tau_1, \ldots, \tau_{i-1}\}$ is periodic from S_{i-1} with a period of H_{i-1}.

Now we consider the scheduling of $\{\tau_1, \ldots, \tau_i\}$, since τ_i is the lowest priority task and since there is no priority inversion with binary delays (Proposition 1) the schedule and thus periodicity of $\{\tau_1, \ldots, \tau_{i-1}\}$ is unchanged by the requests of task τ_i. S_i is the first release of task τ_i after (or at) $S_{i-1} (S_i \geq S_{i-1})$, combining the periodicity of the schedule of $\{\tau_1, \ldots, \tau_{i-1}\}$ and the periodicity of τ_i (from S_i with a period of T_i) we conclude that $\mathcal{P}(i)$ is true: if $\{\tau_1, \ldots, \tau_i\}$ is schedulable, its FTP schedule is periodic from S_i with a period of $\text{lcm}\{H_{i-1}, T_i\} = \text{lcm}\{T_j \mid j = 1, \ldots, i\} = H_i$. □

6 DISCUSSION

In this section, we open the discussion on other properties that could be brought by considering binary preemption delays. We review ways to reduce the pessimism of our simulation interval for any deterministic and memoryless scheduler. Finally, we discuss the future work needed to evolve our model to take into account other interference sources from the platform on the scheduler.

6.1 C-sustainability

We believe that the absence of priority inversion will lead to other positive and better results for the particular case of binary delays. We conjecture that work-conserving priority driven scheduler are C-sustainable [9] for binary delays. The later property means that considering the worst-case execution requirement (C_i) for each task is actually the worst-case regarding schedulability. Consequently, in this case the simulation interval is equivalent to an *exact* simulation based schedulability test. An open question here is whether the assumption of binary delays is realistic or not? To go further, we could also imagine systems where this hypothesis is forced by design when choosing the clock frequencies on an embedded board, for example, or by the tick value of the scheduler if it is a tick scheduling type.

6.2 Reducing the pessimism of the general result

The general result, the simulation interval $[0, H \cdot (n+1) \cdot (\alpha_{\max} + 1) \prod_{i=1}^{n}((O_i + D_i - T_i)_0 + 1))$, can be computed very easily (linear complexity), but is large and pessimistic. This is probably the price to pay for such a general result which is applicable for all schedulers one can imagine (but deterministic and memoryless). Please notice that the scheduler is not necessarily preemptive, not necessarily work-conserving, not necessarily "real-time"…

In order to reduce the size/pessimism of the interval we could reduce the parameters of the system state. In particular, we believe that many real-time schedulers (e.g., EDF, RM, DM, FTP) do not need the totality of the parameters. For example EDF does not need the r parameter because priorities do not change during a reload activity.

6.3 Extending the model

To extend this work, we will also need to consider slightly different models where sections corresponding to the return of a preemption remain preemptive, but must be completely re-executed if preempted. We will also need to extend this property to the other components considered in Section 2.2 in order to model not only preemptions but more generally the scheduling cost of the platform/operating system. In such case, we could also consider a task model with deadline prior to completion as proposed in [10]. Finally, a future extension will be to consider multiprocessor platforms.

7 CONCLUSION

In this paper we have considered the notion of simulation interval for asynchronous arbitrary deadline tasks *with* preemption delays. We have designed and proved correct such an interval for any deterministic and memoryless scheduler upon uniprocessor. We then studied particular cases where significantly shorter simulation intervals can be designed for popular real-time schedulers. Firstly, for EDF we extended the work of Leung and Merril, secondly for FTP we extended the work [17], in both cases for binary delays and constrained deadlines. We also discussed the scope of the results, paths for reducing pessimism as well as other potential results and future work opened by this one.

REFERENCES

[1] Sebastian Altmeyer, Robert I Davis, and Claire Maiza. 2012. Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. *Real-Time Systems* 48, 5 (2012), 499–526.

[2] N. C. Audsley. 1991. *Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times.* Technical Report YCS-164. Department of Computer Science, University of York.

[3] Benjamin Bado, Laurent George, Pierre Courbin, and Joël Goossens. 2012. A semi-partitioned approach for parallel real-time scheduling, In 20th International Conference on Real-Time and Network Systems. *ACM International Conference Proceeding Series*, 151–160. https://doi.org/10.1145/2392987.2393006

[4] Julie Baro, Frédéric Boniol, Mikel Cordovilla, Eric Noulard, and Claire Pagetti. 2012. Off-Line (Optimal) Multiprocessor Scheduling of Dependent Periodic Tasks. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing* (Trento, Italy) *(SAC '12).* Association for Computing Machinery, New York, NY, USA, 1815–1820. https://doi.org/10.1145/2245276.2232071

[5] Sanjoy Baruah. 2005. The limited-preemption uniprocessor scheduling of sporadic task systems. In *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*. 137–144. https://doi.org/10.1109/ECRTS.2005.32

[6] Bernard Berthomieu, P-O Ribet, and François Vernadat. 2004. The tool TINA–construction of abstract state spaces for Petri nets and time Petri nets. *International journal of production research* 42, 14 (2004), 2741–2756.

[7] F. Bimbard. 2007. *Dimensionnement temporel de systèmes embarqués : application à OSEK*. Ph. D. Dissertation.

[8] Alan Burns. 1994. Preemptive Priority Based Scheduling: An Appropriate Engineering Approach. In *Principles of Real-Time Systems*. Prentice Hall, 225–248.

[9] Alan Burns and Sanjoy Baruah. 2008. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering* 2, 1 (2008), 74–97.

[10] A. Burns, K. Tindell, and A.J. Wellings. 1994. Fixed priority scheduling with deadlines prior to completion. In *Proceedings Sixth Euromicro Workshop on Real-Time Systems*. 138–142. https://doi.org/10.1109/EMWRTS.1994.336852

[11] A. Burns, K. Tindell, and A. Wellings. 1995. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Transactions on Software Engineering* 21, 5 (1995), 475–480. https://doi.org/10.1109/32.387477

[12] J.V. Busquets-Mataix, J.J. Serrano, R. Ors, P. Gil, and A. Wellings. 1996. Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In *Proceedings Real-Time Technology and Applications*. 204–212. https://doi.org/10.1109/RTTAS.1996.509537

[13] Annie Choquet-Geniet and Emmanuel Grolleau. 2004. Minimal Schedulability Interval for Real Time Systems of Periodic Tasks with Offsets. *Theoretical Computer Science* 310, 1-3 (2004), 117–134. https://doi.org/10.1016/S0304-3975(03)00362-1

[14] Liliana Cucu and Joël Goossens. 2006. Feasibility Intervals for Fixed-Priority Real-Time Scheduling on Uniform Multiprocessors. In *2006 IEEE Conference on Emerging Technologies and Factory Automation*. IEEE, 397–404. https://doi.org/10.1109/ETFA.2006.355388

[15] Liliana Cucu and Joël Goossens. 2007. Feasibility Intervals for Multiprocessor Fixed-Priority Scheduling of Arbitrary Deadline Periodic Systems. In *2007 Design, Automation Test in Europe Conference Exhibition*. 1–6. https://doi.org/10.1109/DATE.2007.364536

[16] Liliana Cucu-Grosjean and Joël Goossens. 2011. Exact Schedulability Tests for Real-Time Scheduling of Periodic Tasks on Unrelated Multiprocessor Platforms. *Journal of Systems Architecture* 57 (05 2011). https://doi.org/10.1016/j.sysarc.2011.02.007

[17] J. Goossens and R. Devillers. 1997. The Non-Optimality of the Monotonic Priority Assignments for Hard Real-Time Offset Free Systems. *Real-Time Syst.* 13, 2 (sep 1997), 107–126. https://doi.org/10.1023/A:1007980022314

[18] J. Goossens and R. Devillers. 1999. Feasibility intervals for the deadline driven scheduler with arbitrary deadlines. In *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No.PR00306)*. IEEE, 54–61. https://doi.org/10.1109/RTCSA.1999.811193

[19] Joël Goossens, Emmanuel Grolleau, and Liliana Cucu-Grosjean. 2016. Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms. *Real-time systems* 52, 6 (2016), 808–832.

[20] Pierre-Emmanuel Hladik. 2018. A brute-force schedulability analysis for formal model under logical execution time assumption. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. ACM, 609–615.

[21] Daniel I. Katcher, Hiroshi Arakawa, and Jay K. Strosnider. 1993. Engineering and analysis of fixed priority schedulers. *IEEE transactions on Software Engineering* 19, 9 (1993), 920–934.

[22] Joumana Lagha, Jean-Luc Béchennec, Sébastien Faucou, and Olivier-H Roux. 2020. Toward an Exact Simulation Interval for Multiprocessor Real-Time Systems Validation. In *VALID 2020, The Twelfth International Conference on Advances in System Testing and Validation Lifecycle (VALID2020 TheTwelfth International Conference on Advances in System Testing andValidation Lifecycle)*. IARIA, Lisbon, Portugal. https://hal-cnrs.archives-ouvertes.fr/hal-03006791

[23] Chang-Gun Lee, Hoosun Hahn, Yang-Min Seo, Sang Lyul Min, Rhan Ha, Seongsoo Hong, Chang Yun Park, Minsuk Lee, and Chong Sang Kim. 1998. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE Trans. Comput.* 47, 6 (1998), 700–713. https://doi.org/10.1109/12.689649

[24] J. P. Lehoczky. 1990. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *Proceedings of the Real-Time Systems Symposium*. Lake Buena Vista, Florida, USA, 201–213.

[25] Joseph Y-T Leung and M.L. Merrill. 1980. A note on preemptive scheduling of periodic, real-time tasks. *Information processing letters* 11, 3 (1980), 115–118.

[26] Joseph Y.-T. Leung and Jennifer Whitehead. 1982. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation* 2, 4 (1982), 237–250. https://doi.org/10.1016/0166-5316(82)90024-4

[27] Chung Laung Liu and James W Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* 20, 1 (1973), 46–61.

[28] Will Lunniss, Sebastian Altmeyer, Claire Maiza, and Robert I Davis. 2013. Integrating cache related pre-emption delay analysis into edf scheduling. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 75–84.

[29] Mitra Nasri, Robert I Davis, and Björn B Brandenburg. 2018. FIFO with offsets: High schedulability with low overheads. In *IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 271–282.

[30] Vincent Nélis, Patrick Meumeu Yomsi, and Joël Goossens. 2013. Feasibility intervals for homogeneous multicores, asynchronous periodic tasks, and FJP schedulers. In *21st International Conference on Real-Time Networks and Systems, RTNS 2013, Sophia Antipolis, France, October 17-18, 2013*, Michel Auguin, Robert de Simone, Robert I. Davis, and Emmanuel Grolleau (Eds.). ACM, 277–286. https://doi.org/10.1145/2516821.2516848

[31] Guillaume Phavorin, Pascal Richard, Joël Goossens, Thomas Chapeaux, and Claire Maiza. 2015. Scheduling with preemption delays: anomalies and issues. In *International Conference on Real-Time and Networks Systems*. ACM, 109–118.

[32] Frank Singhoff. 2008. About Real Time Scheduling Analysis of Ada Applications. In *Tutorial presented in the 13th International Conference on Reliable Software Technologies, Ada-Europe*.

[33] Frank Singhoff, Jerome Legrand, L Nana Tchamnda, and L Marcé. 2004. Cheddar: a flexible real time scheduling framework. ACM Ada Letters journal, 24 (4): 1-8. In *Also published in the proceedings of the International ACM SIGAda Conference, Atlanta, USA*.

[34] Jan Staschulat and Rolf Ernst. 2005. Scalable precision cache analysis for preemptive scheduling. In *Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*. 157–165.

[35] H. Tomiyama and N.D. Dutt. 2000. Program path analysis to bound cache-related preemption delay in preemptive real-time systems. In *Proceedings of the Eighth International Workshop on Hardware/Software Codesign. CODES 2000 (IEEE Cat. No.00TH8518)*. 67–71.

[36] Hai Nam Tran, Stéphane Rubini, Jalil Boukhobza, and Frank Singhoff. 2021. Feasibility interval and sustainable scheduling simulation with CRPD on uniprocessor platform. *Journal of Systems Architecture* 115 (2021), 102007. https://doi.org/10.1016/j.sysarc.2021.102007

[37] Yun Wang and M. Saksena. 1999. Scheduling fixed-priority tasks with preemption threshold. In *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No.PR00306)*. 328–335. https://doi.org/10.1109/RTCSA.1999.811269

[38] Jia Xu and David Lorge Parnas. 2000. Priority scheduling versus pre-run-time scheduling. *Real-time systems* 18, 1 (2000), 7–23.

[39] Patrick Meumeu Yomsi and Yves Sorel. 2007. Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems. In *Euromicro Conference on Real-Time Systems*. IEEE, 280–290.