A Unifying Response Time Analysis Framework for Dynamic Self-Suspending Tasks

Jian-Jia Chen

Geoffrey Nelissen*

Wen-Hung Huang

*CISTER Research Center

CISTER-TR-160609

2016/07/06

A Unifying Response Time Analysis Framework for Dynamic Self-Suspending Tasks

Jian-Jia Chen, Geoffrey Nelissen*, Wen-Hung Huang

*CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: grrpn@isep.ipp.pt

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

Abstract

For real-time embedded systems, self-suspending behaviors can cause substantial performance/schedulability degradations. In this paper, we focus on preemptive fixed-priority scheduling for the dynamic self-suspension task model on uniprocessor. This model assumes that a job of a task can dynamically suspend itself during its execution (for instance, to wait for shared resources or access co-processors or external devices). The total suspension time of a job is upper-bounded, but this dynamic behavior drastically influences the interference generated by this task on lower-priority tasks. The state-of-the-art results for this task model can be classified into three categories (i) modeling suspension as computation, (ii) modeling suspension as release jitter, and (iii) modeling suspension as a blocking term. However, several results associated to the release jitter approach have been recently proven to be erroneous, and the concept of modeling suspension as blocking was never formally proven correct. This paper presents a unifying response time analysis framework for the dynamic self-suspending task model. We provide a rigorous proof and show that the existing analyses pertaining to the three categories mentioned above are analytically dominated by our proposed solution. Therefore, all those techniques are in fact correct, but they are inferior to the proposed response time analysis in this paper. The evaluation results show that our analysis framework can generate huge improvements (an increase of up to 50% of the number of task sets deemed schedulable) over these state-of-the-art analyses.

A Unifying Response Time Analysis Framework for Dynamic Self-Suspending Tasks

Jian-Jia Chen*, Geoffrey Nelissen§, Wen-Hung Huang*
* TU Dortmund University, Germany
§ CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
Emails: {jian-jia.chen, wen-hung.huang}@tu-dortmund.de, grrpn@isep.ipp.pt

***Abstract*—For real-time embedded systems, self-suspending behaviors can cause substantial performance/schedulability degradations. In this paper, we focus on preemptive fixed-priority scheduling for the dynamic self-suspension task model on uniprocessor. This model assumes that a job of a task can dynamically suspend itself during its execution (for instance, to wait for shared resources or access co-processors or external devices). The total suspension time of a job is upper-bounded, but this dynamic behavior drastically influences the interference generated by this task on lower-priority tasks. The state-of-the-art results for this task model can be classified into three categories (i) modeling suspension as computation, (ii) modeling suspension as release jitter, and (iii) modeling suspension as a blocking term. However, several results associated to the release jitter approach have been recently proven to be erroneous, and the concept of modeling suspension as blocking was never formally proven correct. This paper presents a unifying response time analysis framework for the dynamic self-suspending task model. We provide a rigorous proof and show that the existing analyses pertaining to the three categories mentioned above are analytically dominated by our proposed solution. Therefore, all those techniques are in fact correct, but they are inferior to the proposed response time analysis in this paper. The evaluation results show that our analysis framework can generate huge improvements (an increase of up to 50% of the number of task sets deemed schedulable) over these state-of-the-art analyses.**

I. Introduction

The periodic/sporadic task model has been recognized as the basic model for real-time systems with recurring executions. The seminal work by Liu and Layland [24] considered the scheduling of periodic tasks and presented the schedulability analyses based on utilization bounds to verify whether the deadlines are met or not. For decades, researchers in real-time systems have devoted themselves to effective design and efficient analyses of different recurrent task models to ensure that tasks can meet their specified deadlines. In most of these studies, *a task usually does not suspend itself.* That is, after a job is released, the job is either executed or stays in the ready queue, but it is not moved to the suspension state. Such an assumption is valid only under the following conditions: (1) the latency of the memory accesses and I/O peripherals is considered to be part of the worst-case execution time of a job, (2) there is no external device for accelerating the computation, and (3) there is no synchronization between different tasks on different processors in a multiprocessor or distributed computing platform.

If a job can suspend itself before it finishes its computation, self-suspension behaviour has to be considered. Due to the interaction with other system components and synchronization, self-suspension behaviour has become more visible in designing real-time embedded systems. Typically, the resulting suspension delays range from a few microseconds (e.g., a write operation on a flash drive [17]) to a few hundreds of milliseconds (e.g., offloading computation to GPUs [18], [26]).

There are two typical models for self-suspending sporadic task systems: 1) the dynamic self-suspension task model, and 2) the segmented self-suspension task model. In the *dynamic* self-suspension task model, e.g., [1], [2], [10], [16], [20], [23], [27], in addition to the worst-case execution time C_i of sporadic task τ_i, we have also the worst-case self-suspension time S_i of task τ_i. In the *segmented* self-suspension task model, e.g., [5], [9], [14], [15], [21], [28], the execution behaviour of a job of task τ_i is specified by interleaved computation segments and self-suspension intervals. From the system designer's perspective, the dynamic self-suspension model provides a simple specification by ignoring the juncture of I/O access, computation offloading, or synchronization. However, if the suspending behaviour can be characterized by using a segmented pattern, the segmented self-suspension task model can be more appropriate.

In this paper, we focus on preemptive fixed-priority scheduling for the dynamic self-suspension task model on a uniprocessor platform. To verify the schedulability of a given task set, this problem has been specifically studied in [1], [2], [16], [20], [27]. The recent report by Chen et al. [11] and the report by Bletsas et al. [4] have shown that several analyses in the state-of-the-art of self-suspending tasks [1], [2], [20], [27] are in fact unsafe. Unfortunately, those misconceptions propagated to several works [6], [7], [13], [19], [22], [30]–[32] analyzing the worst-case response time for partitioned multiprocessor real-time locking protocols. Moreover, Liu and Chen in [23] provided a utilization-based schedulability test based on a hyperbolic-form. Huang et al. [16] explored the priority assignment under the same system model.

Furthermore, one result presented by Jane W. S. Liu in her book "Real-Time Systems" [25, p. 164-165] and implicitly used by Rajkumar, Sha, and Lehoczky [29, p. 267] for analyzing the self-suspending behaviour due to synchronization protocols in multiprocessor systems, was never proven correct.

Contributions. The contributions of this paper are as follows:

- We provide a new response analysis framework for dynamic self-suspending sporadic real-time tasks on a uniprocessor platform. The key observation is that the *interference from higher-priority self-suspending tasks can be arbitrarily modelled as jitter or carry-in terms.*
- We prove that the new analysis analytically dominates all the state-of-the-art results, excluding the flawed ones.
- We prove the correctness of the analysis initially proposed in [25, p. 164-165] and [29, p. 267], which were never proven correct in the state-of-the-art[1].

[1]A simplified version of the proof of Theorem 1 to support the correctness of [25, p. 164-165] and [29, p. 267] is provided in [8].

- The evaluation results presented in Section VIII show the huge improvement (an increase of up to 50% of the number of task sets that are deemed schedulable) over the state-of-the-art.

II. Task Model

We assume a system τ composed of n sporadic self-suspending tasks. A sporadic task τ_i is released repeatedly, with each such invocation called a job. The j^{th} job of τ_i, denoted by $\tau_{i,j}$, is released at time $r_{i,j}$ and has an absolute deadline at time $d_{i,j}$. Each job of task τ_i is assumed to have a worst-case execution time C_i. Furthermore, a job of task τ_i may suspend itself for at most S_i time units (across all of its suspension phases). When a job suspends itself, it releases the processor and another job can be executed. The response time of a job is defined as its finishing time minus its release time. Successive jobs of the same task have to execute in sequence.

Each task τ_i is characterized by the tuple (C_i, S_i, D_i, T_i), where T_i is the period (or minimum inter-arrival time) of τ_i and D_i is its relative deadline. T_i specifies the minimum time between two consecutive job releases of τ_i, while D_i defines the maximum amount of time a job can take to complete its execution after its release. It results that for each job $\tau_{i,j}$, $d_{i,j} = r_{i,j} + D_i$ and $r_{i,j+1} \geq r_{i,j} + T_i$. In this paper, we focus on constrained-deadline tasks, for which $D_i \leq T_i$. The utilization of a task τ_i is defined as $U_i = C_i/T_i$.

The worst-case response time (WCRT) R_i of a task τ_i is the maximum response time among all its jobs. A schedulability test for a task τ_k is therefore to verify whether its worst-case response time is no more than its relative deadline D_k. In this paper, we only consider *preemptive fixed-priority scheduling running on a single processor platform*, in which each task is assigned with a unique priority level. We assume that the priority assignment is given beforehand and that the tasks are numbered in a decreasing priority order. That is, a task with a smaller index has a higher priority than any task with a higher index, i.e., task τ_i has a higher-priority than task τ_j if $i < j$.

When performing the schedulability analysis of a specific task τ_k, we will implicitly assume that all the higher priority tasks (i.e., $\tau_1, \tau_2, \ldots, \tau_{k-1}$) are already verified to meet their deadlines, i.e., that $R_i \leq D_i, \forall \tau_i \mid 1 \leq i \leq k-1$.

III. Background

To analyze the worst-case response time (or the schedulability) of a task τ_k, one usually needs to quantify the worst-case interference exerted by the higher-priority tasks on the execution of any job of task τ_k. In the ordinary sequential sporadic real-time task model, i.e., when $S_i = 0$ for every task τ_i, the so-called critical instant theorem by Liu and Layland [24] is commonly adopted. That is, the worst-case response time of task τ_k (if it is less than or equal to its period) happens for the first job of task τ_k when (i) τ_k and all the higher-priority tasks release their first jobs synchronously and (ii) all their subsequent jobs are released as early as possible (i.e., with a rate equal to their periods). However, this definition of the critical instant does not hold for self-suspending tasks.

The analysis of self-suspending task systems requires to model the self-suspending behavior of both the task τ_k under analysis and the higher priority tasks that interfere with τ_k. The techniques employed to model the self-suspension are usually different for τ_k and the higher priority tasks. The worst-case for τ_k happens when its jobs suspend whenever there is no higher-priority job in the system. The resulting behavior is therefore similar as if the suspension time S_k of task τ_k was converted into computation time (see [16] for more detailed explanations). Second, for the higher-priority tasks, we need to consider the self-suspension behaviour that may result in the largest possible interference for task τ_k. There exist three approaches in the state-of-the-art that are potentially sound to perform the schedulability analysis of self-suspending tasks:

- modeling the suspension as execution, also known as the suspension-oblivious analysis (see Section III-A);
- modeling the suspension as release jitter (see Section III-B);
- modeling the suspension as blocking time (see Section III-C).

We later prove in Section VI that all these approaches are analytically correct.

A. Suspension-Oblivious Analysis

The simplest analysis consists in converting the suspension time S_i of each task τ_i as a part of its computation time. Therefore, a constrained-deadline task τ_k can be feasibly scheduled by a fixed-priority scheduling algorithm if

$$\exists t \mid 0 < t \leq D_k, \quad C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil (C_i + S_i) \leq t. \quad (1)$$

B. Modeling the Suspension as Release Jitter

Another approach consists in modeling the impact of the self-suspension S_i of each higher priority task τ_i as release jitter. Several works in the state-of-the-art [1], [2], [20], [27] upper bounded the release jitter with S_i. However, it has been recently shown in [4] that this upper bound is unsafe and the release jitter of task τ_i can in fact be larger than S_i.

Nevertheless, it was proven in the same document [4] that the jitter of a higher-priority task τ_i can be safely upper bounded by $R_i - C_i$. It results that a task τ_k with a constrained deadline can be feasibly scheduled under fixed-priority if

$$\exists t \mid 0 < t \leq D_k, \quad C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + R_i - C_i}{T_i} \right\rceil C_i \leq t. \quad (2)$$

C. Modeling the Suspension as Blocking Time

In [25, p. 164-165], Liu proposed a solution to study the schedulability of a self-suspending task τ_k by modeling the extra delay suffered by τ_k due to the self-suspension behavior of each task in τ as a blocking time.[2] This blocking time has been defined as follows:

- The blocking time contributed from task τ_k is S_k.
- A higher-priority task τ_i can block the execution of task τ_k for at most $\min(C_i, S_i)$ time units.

An upper bound on the blocking time is therefore given by: $B_k = S_k + \sum_{i=1}^{k-1} \min(C_i, S_i)$. In [25], the blocking time is then used to derive a utilization-based schedulability test for rate-monotonic scheduling. Namely, it is stated that, if $T_i = D_i$ for every task $\tau_i \in \tau$ and $\frac{C_k + B_k}{T_k} + \sum_{i=1}^{k-1} U_i \leq k(2^{\frac{1}{k}} - 1)$, then τ_k can be feasibly scheduled with rate-monotonic scheduling.

The same concept was also implicitly used by Rajkumar, Sha, and Lehoczky in [29, p. 267] for analyzing the impact of the self-suspension of a task due to the utilization

[2]Even though the authors in this paper are able to provide a proof to support the correctness, the authors are not able to provide any rationale behind this method which treats suspension time as blocking time.

of synchronization protocols in multiprocessor systems. (See Appendix for details.) If the above argument is correct, we can further prove that a constrained-deadline task τ_k can be feasibly scheduled under fixed-priority scheduling if

$$\exists t \mid 0 < t \leq D_k, \quad C_k + B_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t. \tag{3}$$

However, there is no proof in [25] nor in [29] to support the correctness of those tests. Therefore, in Section VI, we provide a proof (see Theorem 4) of the correctness of Equation (3).

IV. Rationale

Even though it can be proven that the response time analysis associated with Eq.(3) dominates the suspension oblivious one (see Lemma 15 in Section VI), none of the analyses presented in Section III dominates all the others. Hence, Eqs. (2) and (3) are incomparable. That is, in some cases Eq. (3) performs better than Eq. (2), while in others Eq. (2) outperforms Eq. (3).

Example 1. *Consider the two tasks $\tau_1 = (4, 5, 10, 10)$ and $\tau_2 = (6, 1, 19, 19)$. The worst-case response time of τ_1 is obviously 9 whatever the analysis employed. However, the upper bound on R_2 obtained with Eq. (2) is 15, while it is 19 with Eq. (3). The solution obtained with Eq. (2) is tighter.*

Now, let us consider one more task $\tau_3 = (4, 0, 50, 50)$. Using Eq. (2), the worst-case response time R_3 of task τ_3 is upper bounded by the smallest $t > 0$ such that $t = 4 + \left\lceil \frac{t+9-4}{10} \right\rceil 4 + \left\lceil \frac{t+15-6}{19} \right\rceil 6$, which turns out to be 42. With Eq. (3) though, $B_3 = 4 + 1 = 5$ and an upper bound on R_3 is given by the smallest $t > 0$ such that $C_3 + B_3 + \sum_{i=1}^{2} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t$. The solution to this last equation is $t = 37$. Therefore, Eq. (3) provides a tighter bound on R_3 than Eq. (2), while the opposite was true for τ_2. □

In addition to the fact that Eqs. (2) and (3) are incomparable, there are task sets for which both equations overestimate the worst-case response time, e.g., in the following example.

Example 2. *Consider the same three tasks as in Example 1. As explained in Section III-B, the extra interference caused by the self-suspending behavior of τ_1 can be safely modeled by a release jitter equal to $R_1 - C_1 = 5$. Similarly, the extra interference caused by the self-suspension of τ_2 can be modeled by a blocking time equal to $\min(C_2, S_2) = 1$ (see Section III-C). Hence, the worst-case response time R_3 of τ_3 is upper bounded by the smallest $t > 0$ such that $t = 4 + 1 + \left\lceil \frac{t+5}{10} \right\rceil 4 + \left\lceil \frac{t}{19} \right\rceil 6$, which turns out to be 33. This bound on R_3 is smaller than the estimates obtained with both Eqs. (2) and (3) (see Example 1).* □

Example 2 shows that a tighter bound on the worst-case response time of a task can be obtained by combining the properties of the analyses discussed in both Section III-B and III-C. Therefore, in this paper, we derive a response time analysis that draws inspiration from both Eqs. (2) and (3), combining the best of each of them. As further proven in Section VI, the resulting schedulability test dominates all the tests discussed in Section III.

V. A Unifying Analysis Framework

In all this section, we implicitly assume that $R_i \leq D_i, \forall \tau_i \mid 1 \leq i \leq k-1$. This assumption is implicitly used as a fact in all the theorems and lemmas. *Therefore, the worst-case response time or the schedulability of task τ_k has to be verified from $k = 1, 2, \ldots, n$. Here we only focus on the analysis of a certain task τ_k, under the assumption that we have already validated that $R_i \leq D_i \leq T_i, \forall \tau_i \mid 1 \leq i \leq k-1$ (by using any method in this section or Section III).* Our key result in this paper are the two following theorems:

Theorem 1. *Suppose that all tasks $\tau_\ell \mid 1 \leq \ell \leq k$ are schedulable, (i.e., $R_\ell \leq T_\ell$). Then, for any arbitrary vector assignment $\vec{x} = (x_1, x_2, \ldots, x_{k-1})$, in which x_i is either 0 or 1, the worst-case response time R_k of τ_k is upper bounded by the minimum t larger than 0 such that*

$$C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_i^{\vec{x}} + (1 - x_i)(R_i - C_i)}{T_i} \right\rceil C_i \leq t \tag{4}$$

where $Q_i^{\vec{x}} \overset{\text{def}}{=} \sum_{j=i}^{k-1} (S_j \times x_j)$.

Theorem 2. *Suppose that τ_k is not schedulable (i.e., $R_k > T_k$). For any arbitrary vector assignment $\vec{x} = (x_1, x_2, \ldots, x_{k-1})$, in which x_i is either 0 or 1, we have $\forall t | 0 < t \leq T_k$,*

$$C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_i^{\vec{x}} + (1 - x_i)(R_i - C_i)}{T_i} \right\rceil C_i > t$$

where $Q_i^{\vec{x}} \overset{\text{def}}{=} \sum_{j=i}^{k-1} (S_j \times x_j)$.

By Theorems 1 and 2, we can directly derive the following schedulability test.

Corollary 1. *If $\forall \tau_i \mid 1 \leq i < k$, $R_i \leq T_i$, and if there is a vector $\vec{x} = (x_1, x_2, \ldots, x_{k-1})$ with $x_i \in \{0, 1\}$, such that*

$$\exists t | 0 < t \leq D_k,$$
$$C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_i^{\vec{x}} + (1 - x_i)(R_i - C_i)}{T_i} \right\rceil C_i \leq t \tag{5}$$

where $Q_i^{\vec{x}} \overset{\text{def}}{=} \sum_{j=i}^{k-1} (S_j \times x_j)$, then the constrained-deadline task τ_k is schedulable under fixed-priority.

Proof: Let t^* be the first positive value of t respecting Eq. (5). By the assumptions stated in the claim, t^* exists and t^* is smaller than or equal to the deadline D_k. By Theorems 1 and 2, t^* exists and is smaller than or equal to D_k only if τ_k is schedulable. ■

The proof of correctness of Theorems 1 and 2, and hence Corollary 1 is provided in Section V-A. Moreover, we will later prove in Section VI, that Corollary 1 in fact dominates all the analyses discussed in Section III.

We now use the same example as in Section IV, to demonstrate how Corollary 1 can be applied.

Example 3. *Consider the same three tasks used in Examples 1 and 2, i.e., $\tau_1 = (4, 5, 10, 10)$, $\tau_2 = (6, 1, 19, 19)$ and $\tau_3 = (4, 0, 50, 50)$. By the analysis in Example 1, R_1 is upper bounded by 9 and R_2 is upper bounded by 15. Let us assume $R_1 = 9$ and $R_2 = 15$ in the rest of this example. There are four possible vector assignments $\vec{x}$ when considering the schedulability of task τ_3 with Corollary 1. The corresponding procedure to use these four vector assignments can be found in Table I. Among the above four cases, the tests in Cases 2 and 4 are the tightest.* □

Note also that the upper bound on R_3 computed in Example 3, is lower than the estimated worst-case response time obtained in Example 2. The response time analysis presented

$\vec{x}$	Case 1: $(0, 0)$	Case 2: $(0, 1)$	Case 3: $(1, 0)$	Case 4:$(1, 1)$
$(Q_1^{\vec{x}}, Q_2^{\vec{x}})$	$(0, 0)$	$(1, 1)$	$(5, 0)$	$(6, 1)$
condition of Eq. (5)	$4 + \left\lceil \frac{t+0+5}{10} \right\rceil 4 + \left\lceil \frac{t+0+9}{19} \right\rceil 6 \leq t$	$4 + \left\lceil \frac{t+1+5}{10} \right\rceil 4 + \left\lceil \frac{t+1+0}{19} \right\rceil 6 \leq t$	$4 + \left\lceil \frac{t+5+0}{10} \right\rceil 4 + \left\lceil \frac{t+0+9}{19} \right\rceil 6 \leq t$	$4 + \left\lceil \frac{t+6+0}{10} \right\rceil 4 + \left\lceil \frac{t+1+0}{19} \right\rceil 6 \leq t$
upper bound of R_3	42	32	42	32

TABLE I: Detailed procedure in Example 3 for deriving the upper bound of R_3, with $R_1 - C_1 = 5$ and $R_2 - C_2 = 9$.

in Corollary 1 is therefore tighter than the simple combination of existing analysis techniques as proposed in Example 2.

A. Proof of Correctness

We now provide the proof to support the correctness of the response time analysis presented in Theorem 1, whatever the binary values used in vector $\vec{x}$. Throughout the proof, we consider any arbitrary assignment $\vec{x}$, in which x_i is either 0 or 1. For the sake of clarity, we classify the $k-1$ higher-priority tasks into two sets: $\mathbf{T}_0$ and $\mathbf{T}_1$. A task τ_i is in $\mathbf{T}_0$ if x_i is 0; otherwise, it is in $\mathbf{T}_1$. Our analysis is also based on very simple properties and lemmas enunciated as follows:

Property 1. *In a preemptive fixed-priority schedule, the lower-priority jobs do not impact the schedule of the higher-priority jobs.*

Lemma 1. *In a preemptive fixed-priority schedule, if the worst-case response time of task τ_i is no more than its period T_i, removing a job of task τ_i does not affect the schedule of any other jobs of task τ_i.*

Proof: The proof is in Appendix. ■

We now present the detailed proof of Theorems 1 and 2 using the properties stated above. Since the proof is quite long, we will also provide examples to demonstrate the key steps in the proof and lemmas to support intermediate results.

Let Ψ be a fixed-priority preemptive schedule of the task system τ. Suppose that a job J_k of task τ_k arrives at time r_k and finishes at time f_k. By the assumption of $R_k \leq T_k$, we have $f_k \leq r_k + R_k \leq r_k + T_k$. We first prove that Eq. (4) gives us a safe upper bound on $f_k - r_k$ for any job J_k in Ψ if $R_k \leq T_k$. The proof is built upon the three following steps:

1) We discard all the jobs that arrive before r_k and do not contribute to the response time of J_k in the schedule Ψ. We follow an inductive strategy by iteratively inspecting the schedule of the higher priority tasks in Ψ, starting with τ_{k-1} until the highest priority task τ_1. At each iteration, a time instant t_j is identified such that $t_j \leq t_{j+1}$ ($1 \leq j < k$). Then, all the jobs of task τ_j released before t_j are removed from the schedule and, if needed, replaced by an artificial job mimicking the interference caused by the residual workload of task τ_j at time t_j.
2) The final reduced schedule is analyzed to characterize important properties of the reduced schedule in Step 1.
3) We then prove that the response time analysis in Eq. (4) is indeed an upper bound on the worst-case response time R_k of τ_k.

Step 1: Reducing the schedule Ψ

Our purpose in this step is to discard all the jobs that arrive before r_k and have no impact on the response time of J_k in the schedule Ψ. During this step, we iteratively build the schedules from Ψ^k to Ψ^1 mentioned above. Based on a given schedule Ψ^{j+1} (with $1 \leq j < k$), we build the fixed-priority schedule Ψ^j such that the response time of J_k remains identical. At each iteration, we define t_j for task τ_j in the schedule Ψ^{j+1} and build Ψ^j by removing all the jobs released by τ_j before t_j. *We then prove that the response time of J_k in the reduced fixed-priority schedule Ψ^1 remains the same as the response time of J_k in the original fixed-priority schedule Ψ.*

Basic step (definition of Ψ^k and t_k):

We define Ψ^k as the schedule in which (i) all high-priority tasks $\tau_1, \ldots, \tau_{k-1}$ release their jobs at the exact same instants as in Ψ, (ii) τ_k releases only one job at time r_k, (iii) the low-priority tasks $\tau_{k+1}, \ldots, \tau_n$ do not release any job, and (iv) all jobs suspend their execution after the exact same execution time as in Ψ. Moreover, since J_k is released at time r_k and does not finish strictly before f_k, the total amount of idle time of the system from r_k to f_k is at most S_k. In the converted schedule Ψ^k, we further convert the idle time as part of the execution time of J_k. After the conversion by considering suspension as computation for job J_k, we know that the worst-case execution time of J_k is upper bounded by $C'_k = S_k + C_k$. As already discussed in Section III, such a conversion has been widely used. For notational brevity, we denote this job J_k as a release of task $\tau'_k = (C_k + S_k, 0, D_k, T_k)$. It is obvious that Ψ^k remains as a fixed-priority schedule.

Lemma 2. *The response time of J_k in Ψ^k is the same as the response time of J_k in Ψ.*

Proof: We know by Property 1 that the lower priority tasks $\tau_{k+1}, \tau_{k+2}, \ldots, \tau_n$ do not impact the response time of J_k. Therefore, not releasing them has no impact on the response time J_k. Moreover, since we assume that the worst-case response time of task τ_k is no more than T_k, Lemma 1 proves that none of the jobs of task τ_k except J_k impacts the schedule of J_k. Since all the other parameters (i.e., releases and suspensions) that may influence the scheduling decisions are kept identical between Ψ and Ψ^k, the response time of J_k in Ψ^k is identical to the response time of J_k in Ψ. ■

To allow the induction defined below, we also define t_k as the release time of J_k (i.e., $t_k \overset{\text{def}}{=} r_k$).

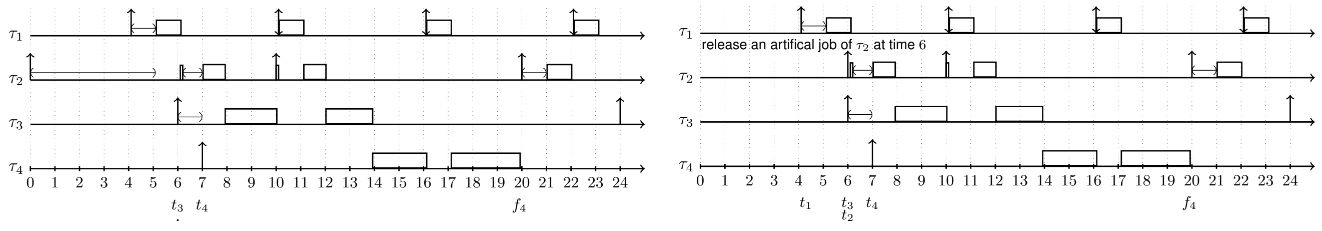
Induction step (definition of Ψ^j and t_j with $1 \leq j < k$):

We define four cases in order to build Ψ^j from Ψ^{j+1}.

Case 0. If all the jobs of task τ_j are released at or after t_{j+1} in schedule Ψ^{j+1}, then we define Ψ^j as being identical to Ψ^{j+1} and set $t_j \overset{\text{def}}{=} t_{j+1}$.

Now, let us consider that task τ_j releases at least one job before t_{j+1} in Ψ^{j+1}. Let r_j be the arrival time of the last job released by τ_j before t_{j+1} in Ψ^{j+1} and let J_j denote that job. By definition, $r_j < t_{j+1}$. Let c_j^* be the remaining execution time of J_j at time t_{j+1} in Ψ^{j+1}. By definition, $0 \leq c_j^* \leq C_j$. In the rest of the proof, c_j^* is called τ_j's *residual workload*.

We start by setting Ψ^j to be identical as Ψ^{j+1}. Then, all the jobs of task τ_j that arrive before r_j are immediately removed from Ψ^j. That is, all jobs released in Ψ^j have identical suspension and execution behavior as in Ψ^{j+1}, and task τ_j



(a) Ψ, Ψ^4 and Ψ^3

(b) Ψ^2 and Ψ^1

Fig. 1: An illustrative example of Step 1 in the proof of Theorem 1 when $\epsilon = 0.1$.

does not release any job before r_j in Ψ^j. There are three cases to decide how we include or exclude J_j in Ψ^j as follows:

Case 1. If $\tau_j \in \mathbf{T}_1$ and J_j does not complete its execution by t_{j+1} in the schedule Ψ^{j+1} (i.e., $c_j^* > 0$), then $t_j \overset{\text{def}}{=} r_j$ and J_j is included in Ψ^j. In this case, task τ_j releases its jobs at exactly the same instants in Ψ^{j+1}, i.e., *at and after* r_j.

Case 2. If $\tau_j \in \mathbf{T}_1$ and J_j completes its execution before or at t_{j+1} in the schedule Ψ^{j+1} (i.e., $c_j^* = 0$), then $t_j \overset{\text{def}}{=} t_{j+1}$ and J_j is further removed and excluded from Ψ^j. In this case, task τ_j releases its jobs at exactly the same instants in Ψ^{j+1} *after* r_j.

Case 3. If $\tau_j \in \mathbf{T}_0$, then $t_j \overset{\text{def}}{=} t_{j+1}$ and (i) τ_j releases its jobs at the same instants in Ψ^{j+1} *after* r_j (i.e., exclude J_j in Ψ^j), and (ii) an artificial (or additional) job J_a with execution time $C_a \overset{\text{def}}{=} c_j^*$ and the same priority as τ_j is released at time t_{j+1}. This artificial job follows the same execution and suspension behavior as job J_j after t_{j+1}.

After the above procedures, it is obvious that the resulting schedule Ψ^j remains as a fixed-priority schedule.

Lemma 3. *The response time of J_k in Ψ^j is the same as the response time of J_k in Ψ^{j+1}.*

Proof: If Case 0 is applied, then Ψ^j and Ψ^{j+1} are identical. The response time of J_k is thus unchanged and the claim trivially holds.

For the rest of the proof, we use the four following facts:

Fact 1. For any ℓ such that $j \leq \ell < k$, there is $t_\ell \leq t_{\ell+1}$ and τ_ℓ does not release any job before t_ℓ in Ψ^j.

Fact 2. No job of $\tau_k, \ldots, \tau_n$ are released before t_k in Ψ^k.

Fact 3. By the assumption that $R_j \leq D_j \leq T_j$ for $j = 1, 2, \ldots, k-1$, removing all the jobs of task τ_j arrived before r_j has no impact on the schedule of any other job released by τ_j (Lemma 1) or any higher priority job released by $\tau_1, \ldots, \tau_{j-1}$ (Property 1). Moreover, by Facts 1 and 2, no task with a priority lower than τ_j (tasks $\tau_{j+1}, \ldots, \tau_n$) release jobs before t_{j+1} in Ψ^{j+1}. Therefore, removing the jobs released and completed by τ_j before r_j does not impact the schedule of the jobs of $\tau_{j+1}, \ldots, \tau_n$. *Consequently, we can safely remove all the jobs of task τ_j arrived and completed before t_{j+1} without impacting the response time of J_k.*

Fact 4. Since by Facts 1 and 2, no task with a priority lower than τ_j (tasks $\tau_{j+1}, \ldots, \tau_n$) releases jobs before t_{j+1} in Ψ^{j+1}, *replacing J_j with the created artificial job (which has the same execution and suspension behavior as J_j from t_{j+1}) has no impact on the schedule of $\tau_{j+1}, \ldots, \tau_n$ in Ψ^{j+1}.*

We now consider the three remaining cases:

In Case 1, Ψ^j is built from Ψ^{j+1} by removing all the jobs released by τ_j *before* r_j. All the excluded jobs have therefore completed their execution at t_{j+1} and by Fact 3, this has no impact on the execution of any job executed after t_{j+1} and thus on the response time of J_k.

In Case 2, Ψ^j is built from Ψ^{j+1} by removing all the jobs released by τ_j before t_{j+1}. Since J_j completes before t_{j+1}, by Fact 3, none of the excluded jobs impacted the response time of J_k. *The response time of J_k in Ψ^j thus remains unchanged in comparison to its response time in Ψ^{j+1}.*

In Case 3, all the jobs of τ_j released before t_{j+1} are removed and the job of task τ_j arrived at time r_j is replaced by a new job released at time t_{j+1} with execution time c_j^* and the same priority as τ_j. *By Facts 3 and 4, the interference generated by τ_j and the additional job on job released at or after t_{j+1} remains identical between Ψ^j and Ψ^{j+1}. Thus, the response time of J_k is unchanged.* ■

Conclusion of Step 1:

This iterative process is repeated until producing Ψ^1. The procedures are well-defined and it is therefore guaranteed that Ψ^1 can be constructed. A pseudo-code of Ψ^1's generation procedure can be found in Appendix. Note that after each iteration, the number of jobs considered in the resulting schedule has been reduced, yet without affecting the response time of J_k, as proven in the following lemma.

Lemma 4. *The response time of job J_k in Ψ^1 is the same as the response time of J_k in Ψ.*

Proof: By Lemma 2, the reponse time of τ_k in Ψ^k is identical to the response time of J_k in Ψ. And by inductively applying Lemma 3, we get that the response time of τ_k in Ψ^k is identical to the response time of τ_k in Ψ^{k-1}, Ψ^{k-2}, …, Ψ^1. This proves the lemma. ■

Example 4. *Consider 4 tasks $\tau_1 = (1, 1, 6, 6)$, $\tau_2 = (1, 6, 10, 10)$, $\tau_3 = (4, 1, 18, 18)$ and $\tau_4 = (5, 0, 20, 20)$. We assume $x_1 = 1$, $x_2 = 0$ and $x_3 = 1$. Figure 1(a) depicts a possible schedule Ψ^4 of those tasks. We assume that the first job of task τ_1 arrives at time $4 + \epsilon$ with $0 < \epsilon < 0.5$. The first job of task τ_2 suspends itself from time 0 to time $5 + \epsilon$, and is blocked by task τ_1 from time $5 + \epsilon$ to time $6 + \epsilon$. After executing ϵ amount of time, the first job of task τ_2 suspends itself again from time $6 + 2\epsilon$ to 7. The schedule in Figure 1(a) is drawn for $\epsilon = 0.1$.*

In the schedule illustrated in Figure 1(a), f_4 is $20 - \epsilon$. Based on the definition of t_k, $t_4 = 7$. Then, we set t_3 to 6 by applying Case 1. The schedule Ψ^3 is identical to the original schedule Ψ^4. When considering task τ_2, we know that J_2 is *the job of task τ_2 arrived at time $r_2 = 0 < t_3$. Since task τ_2 belongs to $\mathbf{T}_0$, by applying Case 3, we have $t_2 = t_3 = 6$ and the residual workload c_2^* is 1. Then, we remove job J_2 from the schedule and create an artificial job with execution time c_2^* that is released at time t_2 and assign the artificial job the same priority level as task τ_2. Note that this artificial job can still suspend itself. Therefore, the schedule Ψ^2, as drawn in Figure 1(b), is slightly different from Ψ^3, shown in Figure 1(a). Yet, the response time of J_4 is unchanged. Finally, t_1 is set to $4+\epsilon$ by applying Case 1 since J_1 (arrived at time $r_1 = 4+\epsilon$) has not completed yet at time $t_2 = 6$. The schedule Ψ^1 is identical to the schedule Ψ^2.* □

Step 2: Analyzing the reduced schedule Ψ^1

We now analyze the properties of the final fixed-priority schedule Ψ^1 in which all the unnecessary jobs have been removed. This step is based on the simple fact that for any interval $[t_1, t)$ with $t \leq f_k$, there is

$$\text{idle}(t_1, t) + \text{exec}(t_1, t) = (t - t_1) \tag{6}$$

where $\text{exec}(t_1, t)$ is the amount of time during which the processor executes tasks within $[t_1, t)$, and $\text{idle}(t_1, t)$ is the amount of time during which the processor remains idle within the interval $[t_1, t)$.

We first provide an upper bound on $\text{idle}(t_1, t)$ (see Lemma 5 and Corollary 2), then on $\text{exec}(t_1, t)$ (see Lemmas 6 to 9). Finally, in Lemma 10, we combine those results with Eq. (6) in order to characterise the schedule Ψ^1 in $[t_1, f_k)$.

We start our analysis with $\text{idle}(t_1, t)$ when $t_1 < t \leq f_k$. Let σ_j be the amount of time during which the processor remains idle within $[t_j, t_{j+1})$ in Ψ^1.

Lemma 5. *For $j = 1, 2, \ldots, k-1$, $\sigma_j = x_j \times \sigma_j \leq x_j \times S_j$.*

Proof: If Case 1 is applied on τ_j when we build Ψ^j in Step 1, (i) $x_j = 1$, (ii) t_j is set to the release time r_j of the job J_j, and (iii) J_j has not completed its execution yet at time t_{j+1}. By (ii) and (iii), the amount of time during which the processor may remain idle within $[t_j, t_{j+1})$ is at most the suspension time S_j of τ_j. Thus, $\sigma_j \leq S_j$. And by (i), $\sigma_j = x_j \times \sigma_j \leq x_j \times S_j$.

If Cases 0, 2 or 3 is applied on τ_j when we build Ψ^j in Step 1, then t_j is equal to t_{j+1} and by definition, $\sigma_j = 0$. It results that $\sigma_j = x_j \times \sigma_j \leq x_j \times S_j$. ■

Corollary 2. *For $i = 1, 2, \ldots, k-1$, $\forall t | t_i < t \leq t_{i+1}$,*

$$\text{idle}(t_1, t) \leq \sum_{j=1}^{i} x_j \sigma_j \leq \sum_{j=1}^{i} x_j S_j \tag{7}$$

Proof: Since $t_i < t \leq t_{i+1}$, $\text{idle}(t_1, t) \leq \sum_{j=1}^{i} \sigma_j$. And by Lemma 5, $\text{idle}(t_1, t) \leq \sum_{j=1}^{i} x_j \sigma_j \leq \sum_{j=1}^{i} x_j S_j$ ■

Example 5. *As shown in the schedule in Example 4, the total idle time from $4+\epsilon$ to $20-\epsilon$, i.e., from $4+\epsilon$ to $5+\epsilon$ and from $6+2\epsilon$ to 7, is $2-2\epsilon$, which is upper-bounded by $S_1 + S_3 = 2$.*

We now consider $\text{exec}(t_1, t)$ when $t_1 < t \leq f_k$. Because there is no job released by lower priority tasks than τ'_k in Ψ^1, we only focus on the execution of the tasks $(\tau_1, \tau_2, \ldots, \tau_{k-1}, \tau'_k)$. Let $\text{exec}_j(t_1, t)$ be *the (accumulative) amount of time that task τ_j is executed in the schedule Ψ^1 in the time interval* $(t_1, t]$. By the construction of the schedule Ψ^1, we know that $\text{exec}_j(t_1, t_j)$ must be equal to 0 since task τ_j is not executed between t_1 and t_j. Therefore, $\text{exec}_j(t_1, t)$ is equal to $\text{exec}_j(t_j, t)$ if $t > t_j$.

Lemma 6. *$\forall t | t_k \leq t < f_k$, the (accumulative) amount of time that task τ'_k is executed from t_k to t is $\text{exec}_k(t_k, t) < C'_k$.*

Proof: Since the finishing time of job J_k is at time f_k in schedule Ψ^1, the condition holds by definition. ■

Lemma 7. *If task $\tau_j \in \mathbf{T}_1$, then $\forall \Delta \geq 0$ we have*

$$\text{exec}_j(t_j, t_j + \Delta) \leq W_j^1(\Delta)$$

where

$$W_j^1(\Delta) \overset{def}{=} \left\lfloor \frac{\Delta}{T_j} \right\rfloor C_j + \min\left\{ \Delta - \left\lfloor \frac{\Delta}{T_j} \right\rfloor T_j, C_j \right\}. \tag{8}$$

Proof: If task $\tau_j \in \mathbf{T}_1$, then Case 0, 1 or 2 is applied when building Ψ^1 in Step 1. In this case, Ψ^1 does not contain any job of task τ_j arrived before t_j (i.e., no residual workload of τ_j at time t_j). Furthermore, $\text{exec}_j(t_j, t_j + \Delta)$ is maximized when the jobs released by τ_j after t_j are actually executing, and hence do not suspend themselves (i.e., τ_j acts as a sporadic tasks without self-suspension). Since, as shown in the literature [3], $W_j^1(\Delta)$, which is usually called *workload function*, is an upper bound on the amount of execution time that a sporadic task can execute without self-suspension, we know that $\text{exec}_j(t_j, t_j + \Delta)$ of τ_j from t_j to $t_j + \Delta$ is upper bounded by $W_j^1(\Delta)$. ■

Lemma 8. *If $\tau_j \in \mathbf{T}_0$, then $\forall \Delta \geq 0$ we have*

$$\text{exec}_j(t_j, t_j + \Delta) \leq \widehat{W}_j^0(\Delta, c_j^*)$$

where

$$\widehat{W}_j^0(\Delta, c_j^*) = \begin{cases} W_j^1(\Delta) & \textit{if } c_j^* = 0 \\ \Delta & \textit{if } c_j^* > 0 \textit{ and } \Delta \leq c_j^* \\ c_j^* & \textit{if } c_j^* > 0 \textit{ and } c_j^* < \Delta \leq \rho_j \\ c_j^* + W_j^1(\Delta - \rho_j) & \textit{if } c_j^* > 0 \textit{ and } \rho_j < \Delta \end{cases} \tag{9}$$

and $\rho_j = (T_j - R_j + c_j^*)$.

Proof: If task $\tau_j \in \mathbf{T}_0$, then Case 0 or 3 is applied when building Ψ^1 in Step 1. Therefore, there might be a job J_j arrived before t_j with a residual workload $0 \leq c_j^* \leq C_j$ at time t_j. The case when $c_j^* = 0$ is identical to the proof of Lemma 7. We now consider the cases where $c_j^* > 0$. Since by assumption $R_j \leq D_j \leq T_j$, task τ_j respects all its deadlines and the worst-case response time, the absolute deadline of the job J_j of τ_j that is not completed yet at t_j, must be at least $t_j + c_j^*$. Therefore, the earliest arrival time of a job of task τ_j *strictly after* t_j is at least $t_j + c_j^* + (T_j - R_j) = t_j + \rho_j$. Since there is no other job of task τ_j released in $[t_j, \rho_j)$ except the artificial job with the residual workload c_j^* created based on J_j, we know that $\text{exec}_j(t_j, t_j + \Delta)$ is upper bounded by $\min\left\{\Delta, c_j^*\right\}$ for $\Delta \leq \rho_j$, thereby proving cases 2 and 3 of Eq. (9). Furthermore, by assumption J_j completes its execution before or at $t_j + \rho_j$. Therefore, following the same proof as Lemma 7, $\text{exec}_j(t_j + \rho_j, t_j + \Delta)$ is upper bounded by $W_j^1(\Delta - \rho_j)$ when $\Delta > \rho_j$. This proves the fourth case of Eq. (9). ■

For notational brevity, let $W_j^0(\Delta) \overset{\text{def}}{=} \widehat{W}_j^0(\Delta, C_j)$. We also prove that, for any $\Delta \geq 0$, $W_j^0(\Delta) \geq \widehat{W}_j^0(\Delta, c_j^*)$:

Lemma 9. *$\forall \Delta \geq 0$, $W_j^0(\Delta) \geq \widehat{W}_j^0(\Delta, c_j^*)$.*

Proof: The proof is based on simple observations of the workload function. The proof is in Appendix. ■

Now that we have derived upper bounds on the idle time $\text{idle}(t_1, t)$ and the execution time $\text{exec}_j(t_j, t_j + \Delta)$ of each task τ_j executed in Ψ^1, we inject those results in Eq. (6) in order to derive properties on the schedule in any interval $[t_1, t)$ for any $t_1 < t < f_k$.

Lemma 10. *$\forall t \mid t_i \leq t < t_{i+1}$ where $i = 1, 2, \ldots, k-1$*

$$\sum_{j=1}^{i} \left(x_j \cdot (W_j^1(t-t_j) + \sigma_j) + (1-x_j) \cdot W_j^0(t-t_j)\right) \geq t - t_1. \tag{10}$$

And, $\forall t \mid t_k \leq t < f_k$,

$$C'_k + \sum_{j=1}^{k-1} \left(x_j \cdot (W_j^1(t-t_j) + \sigma_j) + (1-x_j) \cdot W_j^0(t-t_j)\right) > t - t_1. \tag{11}$$

Proof: We combine the three following facts:

1. By Eq. (6), $\mathrm{idle}(t_1, t) + \mathrm{exec}(t_1, t) = t - t_1$.

2. By Corollary 2, $\mathrm{idle}(t_1, t) \leq \sum_{j=1}^{i-1} x_j \sigma_j$ for all $t \mid t_i \leq t < t_{i+1}$ and $i = 1, 2, \ldots, k-1$.[3]

3. By the construction of the schedule Ψ^1, we know that $\mathrm{exec}_j(t_1, t_j) = 0$ since task τ_j is not executed between t_1 and t_j. Therefore, $\mathrm{exec}_j(t_1, t) = 0$ if $t < t_j$ and $\mathrm{exec}_j(t_1, t) = \mathrm{exec}_j(t_j, t)$ if $t > t_j$. Since $x_j = 0$ if $\tau_j \in \mathbf{T}_0$ and $x_j = 1$ if $\tau_j \in \mathbf{T}_1$, by Lemmas 7, 8 and 9, we have for all $t \mid t_i \leq t < t_{i+1}$ and $i = 1, 2, \ldots, k-1$,

$$\begin{aligned}\mathrm{exec}(t_1, t) &= \sum_{j=1}^{i} \mathrm{exec}_j(t_1, t) = \sum_{j=1}^{i} \mathrm{exec}_j(t_j, t) \\ &\leq \sum_{j=1}^{i} \left(x_j \cdot W_j^1(t-t_j) + (1-x_j) \cdot \widehat{W}_j^0(t-t_j, c_j^*)\right) \\ &\leq \sum_{j=1}^{i} \left(x_j \cdot W_j^1(t-t_j) + (1-x_j) \cdot W_j^0(t-t_j)\right)\end{aligned} \tag{12}$$

Therefore, combining Corollary 2, Eq. (12) and Eq. (6), we obtain Eq. (10).

Moreover, since τ'_k does not complete its execution *strictly* before f_k and because, by definition, τ'_k does not self-suspend, we also know that $\mathrm{idle}(t_k, t) = 0$ for $t_k \leq t < f_k$. Therefore, using Corollary 2, we get for all $t \mid t_k \leq t < f_k$

$$\mathrm{idle}(t_1, t) \leq \sum_{j=1}^{k-1} x_j \sigma_j. \tag{13}$$

Furthermore, by Lemma 6, $\mathrm{exec}_k(t_k, t) < C'_k$ for $t < f_k$. Therefore, adding $\mathrm{exec}_k(t_k, t)$ to Eq. (12), we get for all $t \mid t_k \leq t < f_k$

$$\mathrm{exec}(t_1, t) < C'_k + \sum_{j=1}^{k-1} \left(x_j \cdot W_j^1(t-t_j) + (1-x_j) \cdot W_j^0(t-t_j)\right). \tag{14}$$

Combining Eqs. (13), (14) and (6), we obtain Eq. (11). ∎

Example 6. *Consider the same 4 tasks as in Example 4, for which a possible schedule was depicted in Figure 1 when ϵ is very close to 0. We have $x_1\sigma_1 = 1$, $x_2\sigma_2 = 0$ and $x_3\sigma_3 = 1 - 2\epsilon$. The corresponding functions $W_1^1(t-t_1)$, $W_2^0(t-t_2)$, $W_3^1(t-t_3)$ are illustrated in Figure 2 when ϵ is close to 0 and $R_2 = 10$. As can be seen in Figure 2, the inequalities of Eqs. (10) and (11) clearly hold.* □

[3]The readers may think of using the condition $\mathrm{idle}(t_1, t) \leq \sum_{j=1}^{i-1} x_j S_j$ in Eq. (7) to replace σ_j with S_j. But, this will create a serious problem in Step 3 later, since we cannot always guarantee that $t_i^* \leq t_i$ for $i = 1, 2, \ldots, k$ in Step 3 if we do so in Step 2. Such a treatment should not be applied at this moment here.

Before moving to Step 3, the following lemma is useful for setting the upper bounds of the workload functions.

Lemma 11. *For any $\Delta > 0$, we have*

$$W_j^1(\Delta) \leq \left\lceil \frac{\Delta}{T_j} \right\rceil C_j \tag{15}$$

$$W_j^0(\Delta) \leq \left\lceil \frac{\Delta + R_j - C_j}{T_j} \right\rceil C_j \tag{16}$$

Proof: The upper bound of $W_j^1(\Delta)$ is trivial. Therefore, we focus on the upper bound of $W_j^0(\Delta)$.

If $0 < \Delta \leq C_j$, then by Eq. (9), $W_j^0(\Delta) = \Delta \leq C_j \leq \left\lceil \frac{\Delta + R_j - C_j}{T_j} \right\rceil C_j$.

If $\Delta > C_j$, then by the third and fourth case of Eq. (9)

$$\begin{aligned}W_j^0(\Delta) &\leq C_j + W_j^1(\Delta - (T_j - R_j + C_j)) \\ &\leq C_j + \left\lceil \frac{\Delta - T_j + (R_j - C_j)}{T_j} \right\rceil C_j = \left\lceil \frac{\Delta + R_j - C_j}{T_j} \right\rceil C_j.\end{aligned}$$

∎

Step 3: Creating a Safe Response-Time Upper Bound

The conditions in Lemma 10 cannot be used directly since the values t_j ($j = 1, 2, \ldots, k$) are unknown in the general case. Therefore, Step 3 constructs a safe response-time analysis based on the conditions specified by Eqs. (10) and (11) in Lemma 10. Our goal in this step is to prove that Eq. (4) in Theorem 1 covers all the cases listed in Lemma 10 for any fixed-priority schedule Ψ^1 generated from schedule Ψ.

Our proof strategy is to first artificially move t_i to t_i^* for $i = 1, 2, \ldots, k$ such that $t_i^* \leq t_i$. We define t_i^* as follows:

$$t_i^* \stackrel{\mathrm{def}}{=} \begin{cases} t_1 & \text{if } i = 1 \\ t_{i-1}^* + x_{i-1} \times \sigma_{i-1} & \text{if } i = 2, 3, \ldots, k \end{cases} \tag{17}$$

and we prove that t_i^* is indeed smaller than or equal to t_i.

Lemma 12. *$t_i^* \leq t_i$ for $i = 1, 2, \ldots, k$.*

Proof: By the definition of σ_i, we know that $\sigma_i \leq t_{i+1} - t_i$ for $i = 1, 2, \ldots, k-1$. Therefore, for $i = 2, 3, \ldots, k$,

$$t_i = t_1 + \sum_{j=1}^{i-1} (t_{j+1} - t_j) \geq t_1 + \sum_{j=1}^{i-1} \sigma_j \geq t_1 + \sum_{j=1}^{i-1} x_j \sigma_j = t_i^*$$

since $x_j \in \{0, 1\}$ for any $j = 1, 2, \ldots, i-1$. Finally, the property trivially holds for $i = 1$. ∎

Lemma 13. *$\forall t \mid t_k^* \leq t < f_k$,*

$$C'_k + \sum_{j=1}^{k-1} x_j \cdot W_j^1(t - t_j^*) + (1-x_j) \cdot W_j^0(t - t_j^*) > t - t_k^*. \tag{18}$$

Proof: Because $t_j \geq t_j^*$ by Lemma 12, we have $\forall \Delta \geq 0$

$$W_j^1(\Delta) \leq W_j^1(\Delta + (t_j - t_j^*)) \tag{19}$$

$$W_j^0(\Delta) \leq W_j^0(\Delta + (t_j - t_j^*)). \tag{20}$$

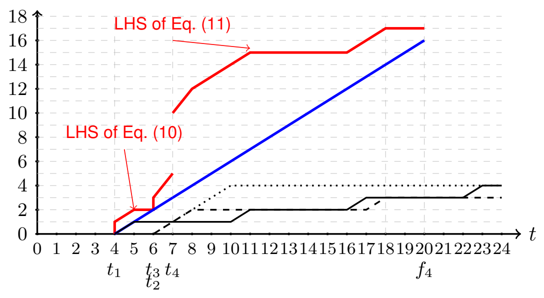


Fig. 2: The workload function for the three higher-priority tasks in Example 4 when ϵ is very close to 0. Solid black line: $W_1^1(t-t_1)$ when $t \geq t_1$, Dashed black line: $W_2^0(t-t_2)$ when $t \geq t_2$, Dotted black line: $W_3^1(t-t_3)$ when $t \geq t_3$, where the three workload functions are 0 if $t-t_j<0$ for $j=1,2,3$, Blue line (the only linear function from $t=4$ in this figure): $t-t_1$, Red line (marked by Eq. (10) and Eq. (11)): left-hand side of Eq. (10) when $t<7$ and left-hand side of Eq. (11) when $7 \leq t < 20$.

It results that, for $j=1,2,\ldots,k-1$, $W_j^1(t-t_j) \leq W_j^1(t-t_j^*)$ and $W_j^0(t-t_j) \leq W_j^0(t-t_j^*)$ for any $t \geq t_j$. Injecting those two inequalities into Eq. (10) $\forall t \mid t_k^* \leq t < t_k$ leads to[4]

$$\sum_{j=1}^{k-1} x_j \cdot (W_j^1(t-t_j^*)+\sigma_j) + (1-x_j)\cdot W_j^0(t-t_j^*) \geq t-t_1,$$

and because by Eq. (17), $t_k^* \overset{\text{def}}{=} t_1 + \sum_{j=1}^{k-1} x_j\sigma_j$, we get

$$\sum_{j=1}^{k-1} x_j \cdot W_j^1(t-t_j^*) + (1-x_j)\cdot W_j^0(t-t_j^*) \geq t-t_k^*, \qquad (21)$$

since $C_k' \geq C_k > 0$, it finally holds that

$$C_k' + \sum_{j=1}^{k-1} x_j \cdot W_j^1(t-t_j^*) + (1-x_j)\cdot W_j^0(t-t_j^*) > t-t_k^*. \quad (22)$$

Similarly, injecting Eqs. (19) and (20) into Eq. (11) $\forall t \mid t_k \leq t < f_k$ leads to

$$C_k' + \sum_{j=1}^{k-1} x_j \cdot W_j^1(t-t_j^*) + (1-x_j)\cdot W_j^0(t-t_j^*) > t-t_k^*. \quad (23)$$

By Eq. (22) (valid for $\forall t \mid t_k^* \leq t < t_k$) and Eq. (23) (valid $\forall t \mid t_k \leq t < f_k$), we prove the lemma. ∎

Lemma 14. *$\forall \theta \mid 0 \leq \theta < f_k - t_k^*$,*

$$C_k' + \sum_{j=1}^{k-1} \left\lceil \frac{\theta + X_j + (1-x_j)(R_j - C_j)}{T_j} \right\rceil C_j > \theta, \qquad (24)$$

where X_j is $\sum_{\ell=j}^{k-1} x_\ell \sigma_\ell$.

Proof: By Eq. (17), we have $t_j^* = t_k^* - \sum_{\ell=j}^{k-1} x_\ell\sigma_\ell$. Therefore, $\forall t \mid t_k^* \leq t < f_k$, we have $t-t_j^* = t-t_k^* + \sum_{\ell=j}^{k-1} x_\ell\sigma_\ell = t-t_k^*+X_j$ for every $j=1,2,\ldots,k-1$. By using Lemma 11 and $t-t_j^*$ above, we can rewrite the condition in Lemma 13 as $C_k' + \sum_{j=1}^{k-1}\left(x_j \left\lceil \frac{t-t_k^*+X_j}{T_j}\right\rceil C_j + (1-x_j)\left\lceil \frac{t-t_k^*+X_j+R_j-C_j}{T_j}\right\rceil C_j\right) > t-t_k^*$, $\forall t \mid t_k^* \leq t < f_k$. Since x_j is either 0 or 1, this is equivalent to $\forall t \mid t_k^* \leq t < f_k$,

$$C_k' + \sum_{j=1}^{k-1} \left\lceil \frac{t-t_k^* + X_j + (1-x_j)(R_j - C_j)}{T_j} \right\rceil C_j > t - t_k^*$$

By replacing $t-t_k^*$ with θ, we reach the conclusion. ∎

[4]This holds since the interval $[t_k^*, t_k]$ is fully covered by the interval $[t_1, t_k]$.

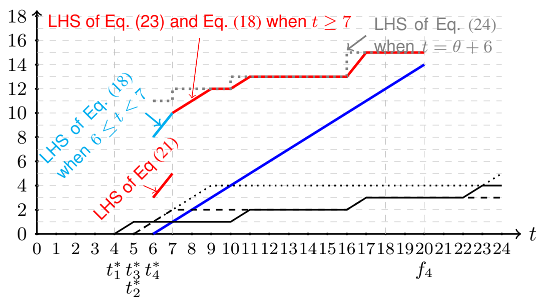


Fig. 3: The workload function for the three higher-priority tasks in Example 4 when ϵ is very close to 0. Solid black line: $W_1^1(t-t_1^*)$ when $t \geq t_1^*$, Dashed black line: $W_2^0(t-t_2^*)$ when $t \geq t_2^*$, Dotted black line: $W_3^1(t-t_3^*)$ when $t \geq t_3^*$, where the three workload functions are 0 if $t-t_j^*<0$ for $j=1,2,3$, Blue line (the only linear function from $t=6$ in this figure): $t-t_k^*=t-6$, Red line (marked by Eq. (21) and Eq. (23)): left-hand side of Eq. (21) when $t<7$ and left-hand side of Eq. (23) and Eq. (18) when $7 \leq t < 20$, Purple line (marked by Eq. (18) when $6 \leq t < 7$), Gray dotted line (marked by Eq. (24)) by setting $\theta = t-6$.

Proof of Theorem 1. The condition in Lemma 14 implies that the minimum θ with $\theta > 0$ and $C_k' + \sum_{j=1}^{k-1}\left\lceil \frac{\theta+X_j+(1-x_j)(R_j-C_j)}{T_j}\right\rceil C_j = \theta$ is larger than or equal to $f_k - t_k^* \geq f_k - t_k$ and therefore provides an upper bound on any job J_k released in any schedule Ψ. However, the condition in Lemma 14 still requires the knowledge of σ_i. Yet, it is straightforward to see that $\sum_{j=1}^{k-1}\left\lceil \frac{\theta+X_j+(1-x_j)(R_j-C_j)}{T_j}\right\rceil C_j$ is maximized when X_j is the largest. Since by Lemma 5 $X_j = \sum_{\ell=j}^{k-1} x_\ell\sigma_\ell \leq \sum_{\ell=j}^{k-1} x_\ell S_\ell = Q_j^{\vec{x}}$, we reach the conclusion of the correctness of Theorem 1 when replacing X_j with $Q_j^{\vec{x}}$ in the previous equation. □

Example 7. *We consider Example 4 when ϵ is very close to 0 to illustrate Step 3 in the proof of Theorem 1. For such a case, $t_1^*=4, t_2^*=5, t_3^*=5$, and $t_4^*=6$. Figure 3 presents the corresponding relations of the inequalities in Step 3. As shown in Figure 3, all the inequalities in Eqs.* (21), (23), (18), *and* (24) *hold.* □

The physical meaning of the above analysis in Theorem 1 can be found in Appendix.

As mentioned at the beginning of this section, all the lemmas and corollaries proven above are valid for any job in any schedule Ψ where all the jobs respect their deadlines. Yet, those results are still valid for the first job of task τ_k that misses a deadline in a schedule Ψ (if such a job exists). This allows us to prove Theorem 2 below.

Proof of Theorem 2. By the assumption that $R_k > T_k$, there exists a schedule Ψ such that the response time of at least one job of τ_k is strictly larger than T_k. Let J_k be the *first* job in the schedule Ψ that has response time larger than T_k. Suppose that J_k arrives at time r_k. When job J_k is released at time r_k, there is no other unfinished job of task τ_k. By Lemma 1, we can safely remove all the other jobs of task τ_k arrived before r_k without affecting the response time of J_k. It is rather straightforward to see that removing all the other jobs of task τ_k arrived after r_k also does not change the fact that J_k finishes after $r_k + T_k$. Let f_k be the time at which J_k finishes in the above schedule after removing the other jobs of task τ_k. We know that $f_k - r_k > T_k$.

Then, we can follow all the procedures and steps in the proof of Theorem 1 to reach the same conclusion in Lemma 14, which implies Theorem 2 by setting $X_j \leq Q_j^{\vec{x}}$ for $j = 1, 2, \ldots, k-1$ since $f_k - r_k > T_k$ and $C'_k = C_k + S_k$. □

VI. Dominance over the State of the Art

In this section, we prove that the schedulability test presented in Corollary 1 dominates all the existing tests in the state-of-the-art, in the sense that if a task set is deemed schedulable by either of the tests presented in Section III, then it is also deemed schedulable by Corollary 1.

Lemma 15. *The schedulability test of task τ_k provided by Eq.* (3) *dominates that of Eq.* (1).

Proof: For any $t > 0$, it is straightforward to see that

$$\begin{aligned}
&C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil (C_i + S_i) \\
\geq & C_k + S_k + \sum_{i=1}^{k-1} S_i + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i \\
\geq & C_k + S_k + \sum_{i=1}^{k-1} \min(C_i, S_i) + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i
\end{aligned}$$

and by using the definition of B_k (i.e., in Section III-C), we get

$$C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil (C_i + S_i) \geq C_k + B_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i$$

Therefore, Eq. (3) will always have a solution which is smaller than or equal to the solution of Eq. (1). This proves the lemma. ■

Lemma 16. *The schedulability test presented in Corollary 1 dominates the schedulability test provided by Eq.* (2).

Proof: Consider the case where $x_1 = x_2 = \cdots = x_{k-1} = 0$. Eq. (5) becomes identical to Eq. (2) for this particular vector assignment. Therefore, if Eq. (2) deems a task set as being schedulable, so does Corollary 1. This proves the lemma. ■

Lemma 17. *The schedulability test presented in Corollary 1 dominates the schedulability test provided by Eq.* (3).

Proof: In this proof, we first transform the worst-case response time analysis presented in Corollary 1 in a more pessimistic analysis. We then prove that this more pessimistic version of Corollary 1 provides the same solution as Eq. (3), which then proves the lemma. Due to space limitation, the proof is in Appendix. ■

Theorem 3. *The schedulability test presented in Corollary 1 dominates the schedulability tests provided by Equations* (1), (2), *and* (3).

Proof: It is a direct application of Lemmas 15, 16 and 17. ■

As a corollary of this theorem, it directly follows that all the response time analyses discussed in Section III are in fact correct. This provides the first proof of correctness for Eq. (3), which was initially presented in [25] but never proven correct.

Theorem 4. *The schedulability tests provided by Eqs* (1), (2), *and* (3) *are all correct.*

Proof: It directly results from the two following facts,

(i) by Theorem 3, the schedulability test presented in Corollary 1 dominates the schedulability tests provided by Equations (1), (2), and (3);
(ii) as proven in Section V-A, Corollary 1 is correct.

■

VII. Linear Approximation

To test the schedulability of a task τ_k, Corollary 1 implies to test all the possible vector assignments $\vec{x} = (x_1, x_2, \ldots, x_{k-1})$ to get the tightest result (under our analysis). Therefore, 2^{k-1} possible combinations should be tested, implying exponential time complexity. In this section, we thus provide a solution to reduce the time complexity associated to Corollary 1. Indeed, using a linear approximation of the test in Eq. (5), a good vector assignment can be derived in linear time.

By the definition of the ceiling operator, it holds that:

$$\begin{aligned}
&C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + \sum_{\ell=i}^{k-1} x_\ell S_\ell + (1 - x_i)(R_i - C_i)}{T_i} \right\rceil C_i \\
\leq & C_k + S_k + \sum_{i=1}^{k-1} \left(\frac{t + \sum_{\ell=i}^{k-1} x_\ell S_\ell + (1 - x_i)(R_i - C_i)}{T_i} + 1 \right) C_i \\
= & C_k + S_k + \sum_{i=1}^{k-1} \left(U_i \cdot t + C_i + U_i(1 - x_i)(R_i - C_i) + U_i \sum_{\ell=i}^{k-1} x_\ell S_\ell \right)
\end{aligned} \tag{25}$$

Moreover, using the simple algebra property that for any two vectors $\vec{a}$ and $\vec{b}$ of size $(k-1)$ there is $\sum_{i=1}^{k-1} a_i \sum_{j=i}^{k-1} b_j = \sum_{j=1}^{k-1} b_j \sum_{i=1}^{j} a_i$, we get that $\sum_{i=1}^{k-1} U_i \sum_{\ell=i}^{k-1} x_\ell S_\ell = \sum_{i=1}^{k-1} x_i S_i \sum_{\ell=1}^{i} U_\ell$. Hence, injecting this last expression in Eq. (25), it holds that

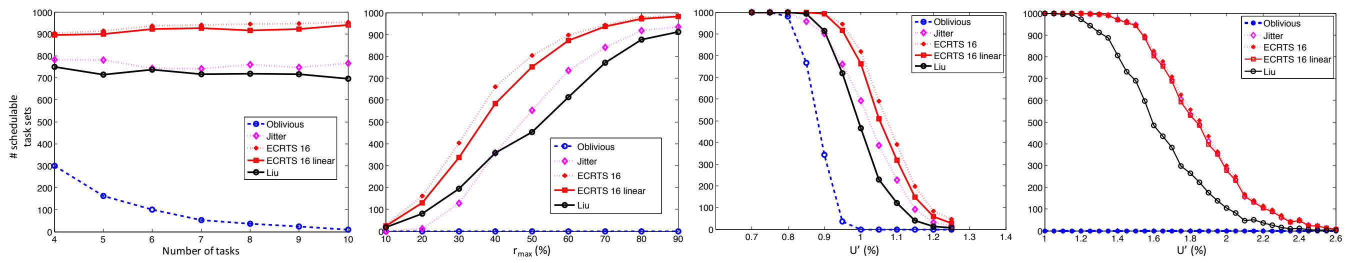
$$\begin{aligned}
&C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + \sum_{\ell=i}^{k-1} x_\ell S_\ell + (1 - x_i)(R_i - C_i)}{T_i} \right\rceil C_i \\
\leq & C_k + S_k + \sum_{i=1}^{k-1} \left(U_i \cdot t + C_i + U_i(1 - x_i)(R_i - C_i) + x_i S_i \sum_{\ell=1}^{i} U_\ell \right)
\end{aligned}$$

It results that the minimum positive value for t such that

$$C_k + S_k + \sum_{i=1}^{k-1} \left(U_i t + C_i + U_i(1 - x_i)(R_i - C_i) + x_i S_i \sum_{\ell=1}^{i} U_\ell \right) \leq t \tag{26}$$

is an upper bound on the worst-case response time R_k of τ_k.

Observing Eq. (26), the contribution of x_i can be individually determined as $U_i(R_i - C_i)$ when x_i is 0 or $S_i(\sum_{\ell=1}^{i} U_\ell)$ when x_i is 1. Therefore, whether x_i should be set to 0 or 1 can be decided by individually comparing the two constants $U_i(R_i - C_i)$ and $S_i(\sum_{\ell=1}^{i} U_\ell)$. Eq. (26) is therefore minimized when $x_i = 1$ if $U_i(R_i - C_i) > S_i(\sum_{\ell=1}^{i} U_\ell)$ and when $x_i = 0$



(a) Varying the number of tasks n, $U' = 0.95$, $r_{\min} = 0.05$, $r_{\max} = 0.5$ (b) Varying $r_{\max}$ (unit: %), $U' = 1$, $n = 8$, $r_{\min} = 0.05$ (c) Varying U', $n = 8$, $r_{\min} = 0.05$, $r_{\max} = 0.5$ (d) Varying U', $n = 8$, $r_{\min} = 0.5$, $r_{\max} = 0.9$

Fig. 4: Number of schedulable task sets over 1000 randomly generated task sets.

otherwise. We denote the resulting vector by $\vec{x}^{lin}$, where, for each higher-priority task τ_i,

$$x_i^{lin} = \begin{cases} 1 & \text{if } U_i(R_i - C_i) > S_i(\sum_{\ell=1}^{i} U_\ell) \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

The following properties directly follow.

Property 2. *For any $t > 0$, the vector assignment $\vec{x}^{lin}$ minimizes the solution to Eq. (26) among all 2^{k-1} possible vector assignments.*

Theorem 5. *Let $rbf_k(t, \vec{x}^{lin})$ be the left hand side of Eq. (26). Task τ_k is schedulable under fixed-priority if*

$$rbf_k(D_k, \vec{x}^{lin}) \leq D_k. \tag{28}$$

Proof: It directly follows from Corollary 1 and the fact that, by construction, Eq. (26) upper bounds Eq. (4). Note that $rbf_k(t, \vec{x}^{lin})$ can be expressed as $A + \sum_{i=1}^{k-1} U_i t$ with a constant $A > 0$ (independent from t). Therefore, if the condition in Eq. (26) holds for a certain $0 < t < D_k$ with $A + \sum_{i=1}^{k-1} U_i t \leq t$, then the inequality $A + \sum_{i=1}^{k-1} U_i D_k \leq D_k$ also holds. ■

Property 3. *The time complexity of both deriving $\vec{x}^{lin}$ and testing Eq. (26) is $O(k)$.*

VIII. Experiments

In this section, we present experiments conducted on randomly generated task sets. Five schedulability tests are compared, namely, the suspension oblivious approach (Section III-A), the modeling of suspension as release jitter (Section III-B), the analysis that models the suspension as a blocking term (Section III-C), the generic framework of Corollary 1 (called ECRTS 16 in the plots) and the schedulability test of Theorem 1 based on the vector defined in Eq. (27) in Section VII (called ECRTS 16 linear in the plots). In those experiments, the tasks are assumed to be scheduled with rate monotonic and have implicit deadlines (i.e., $D_i = T_i$).

The task sets were generated using the `randfixedsum` algorithm presented in [12]. Let C'_i denote the sum of C_i and S_i (i.e., $C'_i \overset{\text{def}}{=} C_i + S_i$). The modified utilization of τ_i is then given by $U'_i \overset{\text{def}}{=} C'_i / T_i$ and the total modified utilization is $U' \overset{\text{def}}{=} \sum_{i=1}^{n} U'_i$. The task generator uses the `randfixedsum` algorithm to generate n values of U'_i (one for each task) with total modified utilization U'. A period T_i is then randomly generated from a uniform distribution spanning from 100 to 10000. The value $C'_i = U'_i \times T_i$ is then divided in the two components C_i and S_i using a random ratio r_i from a uniform distribution between a value $r_{\min}$ and $r_{\max}$ depending of the specific experiment performed. That is, $S_i \overset{\text{def}}{=} r_i \times C'_i$ and $C_i = (1 - r_i) \times C'_i$. Each point in the plots of Figure 4 represents the number of task sets that were deemed schedulable by the respective algorithm over 1000 experiments.

Four different types of experiments are reported in this paper. The first one is illustrated in Figure 4a. It presents the evolution of the number of task sets deemed schedulable when the number of self-suspending tasks increases. The number of tasks n is varied from 4 to 10 for a total modified utilization U' of 0.95. As can be seen in Figure 4a, at the exception of the suspension oblivious analysis, the performance of the tests is barely influenced by the number of tasks. In fact, the number of task sets found schedulable by the test of Corollary 1 and the linear test of Section VII slightly increases with the number of tasks. It is the opposite behavior in the suspension oblivious approach. One can already conclude from this plot that the tests developed in this paper perform way better than the state-of-the-art. Furthermore, the difference between the performance of Corollary 1 and its linear version is quite small, thereby making the linear test a practical and useful analysis.

The second experiment is presented in Figure 4b and shows the evolution of the performance of the tests with respect to the length of the total suspension time of a task when the total modified utilization U' and the number of tasks are kept constant. The value of $r_{\max}$ is then varied from 10% to 90%, hence increasing the number of tasks with high suspension times. The value $r_{\min}$ is kept constant at 5%, so as to keep a certain diversity in the suspension behavior of each task. As expected, the suspension oblivious approach does not accept any task set since the total modified utilization is equal to 100%. For the other tests however, the number of schedulable task sets increases when the suspension times become larger. Indeed, the actual workload, which accounts only for the WCET C_i, decreases when S_i increases. Again, one can see the improvement of the tests of this paper over the state-of-the-art. Interestingly, one can also witness the incomparability of the jitter-based and the blocking based schedulability tests.

The last two plots (Figures 4c and 4d), present the results obtained when the total modified utilization increases but the distribution of suspension times and the number of tasks remain identical. As expected, the number of schedulable task sets decreases when the utilization increases. The improvement of Corollary 1 over the state-of-the-art is still high when suspension times are in average smaller than the execution times of the tasks (see Figures 4c). However, when the suspension time becomes larger than the execution time of the task (see Figures 4d), the release jitter-based test performs almost as well as Corollary 1 since the best vector assignment is usually to set all the x_i to 0 for such cases.

IX. Conclusion

In this paper, we studied the preemptive fixed-priority scheduling of dynamic self-suspending tasks running on a uniprocessor platform. This paper presents a unifying response time analysis framework in Theorems 1 2 and Corollary 1. We show that this result analytically dominates all the existing analyses presented in Section III, and, by doing such, we also implicitly proved the correctness of all these analyses. Although Corollary 1 requires exponential time complexity, we show that a simpler algorithm presented in Section VII can help accelerate the analysis while outputting good results.

Acknowledgements. The authors would like to thank the anonymous reviewers for their suggestions and helps to improve the presentation flow and the clarity of the proof of Theorem 1.

This paper is supported by DFG, as part of the Collaborative Research Center SFB876 (http://sfb876.tu-dortmund.de/). This work was also partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER); also by FCT/MEC and the EU ARTEMIS JU within project(s) ARTEMIS/0003/2012 - JU grant nr. 333053 (CONCERTO) and ARTEMIS/0001/2013 - JU grant nr. 621429 (EMC2).

References

[1] N. C. Audsley and K. Bletsas. Fixed priority timing analysis of real-time systems with limited parallelism. In *16th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 231–238, 2004.

[2] N. C. Audsley and K. Bletsas. Realistic analysis of limited parallel software / hardware implementations. In *10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 388–395, 2004.

[3] M. Bertogna, M. Cirinei, and G. Lipari. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In *Principles of Distributed Systems*, pages 306–321. Springer, 2006.

[4] K. Bletsas, N. Audsley, W.-H. Huang, J.-J. Chen, and G. Nelissen. Errata for three papers (2004-05) on fixed-priority scheduling with self-suspensions. Technical Report CISTER-TR-150713, CISTER, July 2015.

[5] K. Bletsas and N. C. Audsley. Extended analysis with reduced pessimism for systems with limited parallelism. In *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 525–531, 2005.

[6] B. Brandenburg. Improved analysis and evaluation of real-time semaphore protocols for P-FP scheduling. In *RTAS*, 2013.

[7] A. Carminati, R. de Oliveira, and L. Friedrich. Exploring the design space of multiprocessor synchronization protocols for real-time systems. *Journal of Systems Architecture*, 60(3):258–270, 2014.

[8] J.-J. Chen, W.-H. Huang, and G. Nelissen. A note on modeling self-suspending time as blocking time in real-time systems. *Computing Research Repository(CoRR)*, 2016. http://arxiv.org/abs/1602.07750.

[9] J.-J. Chen and C. Liu. Fixed-relative-deadline scheduling of hard real-time tasks with self-suspensions. In *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium (RTSS)*, pages 149–160, 2014. **A typo in the schedulability test in Theorem 3 was identified on 13, May, 2015**. http://ls12-www.cs.tu-dortmund.de/daes/media/documents/publications/downloads/2014-chen-FRD-erratum.pdf.

[10] J.-J. Chen, G. Nelissen, and W.-H. Huang. A unifying response time analysis framework for dynamic self-suspending tasks. Technical Report 850, Faculty of Informatik, TU Dortmund, 2016.

[11] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, Neil, Audsley, R. Rajkumar, and D. de Niz. Many suspensions, many problems: A review of self-suspending tasks in real-time systems. Technical Report 854, Faculty of Informatik, TU Dortmund, 2016.

[12] P. Emberson, R. Stafford, and R. I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *WATERS Workshop*, pages 6–11, 2010.

[13] G. Han, H. Zeng, M. Natale, X. Liu, and W. Dou. Experimental evaluation and selection of data consistency mechanisms for hard real-time applications on multicore platforms. *IEEE Transactions on Industrial Informatics*, 10(2):903–918, 2014.

[14] W.-H. Huang and J.-J. Chen. Schedulability and priority assignment for multi-segment self-suspending real-time tasks under fixed-priority scheduling. Technical report, Dortmund, Germany, 2015.

[15] W.-H. Huang and J.-J. Chen. Self-suspension real-time tasks under fixed-relative-deadline fixed-priority scheduling. In *Design, Automation, and Test in Europe (DATE)*, 2016.

[16] W.-H. Huang, J.-J. Chen, H. Zhou, and C. Liu. PASS: Priority assignment of real-time tasks with dynamic suspending behavior under fixed-priority scheduling. In *Proceedings of the 52nd Annual Design Automation Conference on - DAC15*. Association for Computing Machinery (ACM), 2015.

[17] W. Kang, S. Son, J. Stankovic, and M. Amirijoo. I/O-Aware Deadline Miss Ratio Management in Real-Time Embedded Databases. In *Proc. of the 28th IEEE Real-Time Systems Symp.*, pages 277–287, 2007.

[18] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar. RGEM: A Responsive GPGPU Execution Model for Runtime Engines. In *2011 IEEE 32nd Real-Time Systems Symposium*, 2011.

[19] H. Kim, S. Wang, and R. Rajkumar. vMPCP: a synchronization framework for multi-core virtual machines. In *RTSS*, 2014.

[20] I. Kim, K. Choi, S. Park, D. Kim, and M. Hong. Real-time scheduling of tasks that contain the external blocking intervals. In *RTCSA*, pages 54–59, 1995.

[21] J. Kim, B. Andersson, D. de Niz, J.-J. Chen, W.-H. Huang, and G. Nelissen. Segment-fixed priority scheduling for self-suspending real-time tasks. Technical Report CMU/SEI-2016-TR-002, CMU/SEI, 2016.

[22] K. Lakshmanan, D. De Niz, and R. Rajkumar. Coordinated task scheduling, allocation and synchronization on multiprocessors. In *RTSS*, 2009.

[23] C. Liu and J.-J. Chen. Bursty-interference analysis techniques for analyzing complex real-time task models. In *Real-Time Systems Symposium (RTSS)*, pages 173–183, 2014.

[24] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, jan 1973.

[25] J. W. S. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[26] W. Liu, J.-J. Chen, A. Toma, T.-W. Kuo, and Q. Deng. Computation Offloading by Using Timing Unreliable Components in Real-Time Systems. In *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference (DAC)*, 2014.

[27] L. Ming. Scheduling of the inter-dependent messages in real-time communication. In *Proc. of the First International Workshop on Real-Time Computing Systems and Applications*, 1994.

[28] J. C. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS)*, pages 26–37, 1998.

[29] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS '88)*, pages 259–269, 1988.

[30] M. Yang, H. Lei, Y. Liao, and F. Rabee. PK-OMLP: An OMLP based k-exclusion real-time locking protocol for multi- GPU sharing under partitioned scheduling. In *DASC*, 2013.

[31] M. Yang, H. Lei, Y. Liao, and F. Rabee. Improved blocking time analysis and evaluation for the multiprocessor priority ceiling protocol. *Journal of Computer Science and Technology*, 29(6):1003–1013, 2014.

[32] H. Zeng and M. Natale. Mechanisms for guaranteeing data consistency and flow preservation in AUTOSAR software on multi-core platforms. In *SIES*, 2011.

APPENDIX

How did Rajkumar, Sha, and Lehoczky in [29, p. 267] analyze dynamic self-suspending behaviour due to multiprocessor synchronization? The statement in [29] reads as follows:

> *"For each higher priority job $\tau_{i,j}$ that suspends on global semaphores or for other reasons, add the term $\min(C_i, S_i)$ to B_k, where S_i is the maximum duration that $\tau_{i,j}$ can suspend itself. [...] The sum [...] yields B_k, which in turn can be used in $\frac{C_k+B_k}{T_k} + \sum_{i=1}^{k-1} U_i \leq k(2^{\frac{1}{k}} - 1)$ to determine whether the current task allocation to the processor is schedulable."*

We rephrased the wording and notation in order to be consistent with this paper. Moreover, the multiprocessor scheduling in such a case is based on partitioned scheduling. Therefore, the schedulability analysis of a task set on a processor is the same as the uniprocessor problem by additionally considering the self-suspending behaviour due to the synchronization with other tasks on other processors.

Proof of of Lemma 1. Since, by assumption, the worst-case response time of task τ_i is no more than its period, any job $\tau_{i,j}$ of task τ_i completes its execution before the release of the next job $\tau_{i,j+1}$. Hence, the execution of $\tau_{i,j}$ does not directly interfere with the execution of any other job of τ_i, which then depends only on the schedule of the higher priority jobs. Furthermore, as stated in Property 1, the removal of $\tau_{i,j}$ has no impact on the schedule of the higher-priority jobs, thereby implying that the other jobs of task τ_i are not affected by the removal of $\tau_{i,j}$. □

Transformation from Ψ to Ψ^1: Here, we present the pseudo-code to transform from the given schedule Ψ to Ψ^1:

Algorithm 1 Transformation from Ψ to Ψ^1

```
Input: $\tau_k$, $\mathbf{T}_0$, $\mathbf{T}_1$, and a fixed-priority preemptive schedule $\Psi$ of $\tau$ under
    the assumption $R_k \leq T_k$;
1:  pick one job $J_k$ of task $\tau_k$ and set $r_k$ as the arrival time of $J_k$;
2:  remove all the jobs generated from $\tau_k, \tau_{k+1}, \tau_{k+2}, \ldots, \tau_n$ in the
    schedule $\Psi$, except $J_k$;
3:  $\Psi^k \leftarrow \Psi$ and $t_k \leftarrow r_k$;
4:  for $j \leftarrow k-1$ to 1 do
5:      let $r_j$ be the arrival time of the last job released by $\tau_j$ before $t_{j+1}$
        in $\Psi^{j+1}$ and let $J_j$ denote that job;
6:      if $r_j$ does not exist then
7:          $\Psi^j \leftarrow \Psi^{j+1}$ and $t_j \leftarrow t_{j+1}$; {Case 0}
8:      else
9:          $\Psi^j \leftarrow \Psi^{j+1}$ and remove all the jobs of task $\tau_j$ released before
            $r_j$ in schedule $\Psi^j$;
10:         if $\tau_j \in \mathbf{T}_0$ then
11:             $t_j \leftarrow t_{j+1}$, remove $J_j$, and create an artificial job to represent
                the residual workload of $J_j$, executed at or after $t_{j+1}$; {Case
                3}
12:         else
13:             if $J_j$ completes its execution at or before $t_{j+1}$ then
14:                 $t_j \leftarrow t_{j+1}$, remove $J_j$ in schedule $\Psi^j$; {Case 2}
15:             else
16:                 $t_j \leftarrow r_j$; {Case 1}
17:             end if
18:         end if
19:     end if
20: end for
21: return $\Psi^1$;
```

Proof of of Lemma 9. We first prove that $\widehat{W}_j^0(\Delta, C_j) \geq \widehat{W}_j^0(\Delta, c_j^*)$ when $c_j^* = 0$. That is, we prove that $\widehat{W}_j^0(\Delta, C_j) \geq W_j^1(\Delta)$ (see Eq. (9)).

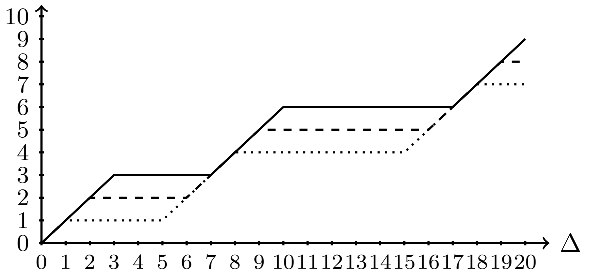


Fig. 5: The workload function $\widehat{W}_j^0(\Delta, c_j^*)$ when $T_j = 10$, $C_j = 3$, and $R_j = 6$. Solid line: c_j^* is 3, Dashed line: c_j^* is 2, Dotted line: c_j^* is 1.

By definition, $\rho_j = T_j - R_j + C_j$ when c_j^* is C_j. Because by assumption $C_j \leq R_j \leq T_j$, we have $0 \leq \rho_j \leq T_j$. Therefore, for $\Delta \geq T_j$, we have $W_j^1(\Delta) = C_j + W_j^1(\Delta - T_j) \leq C_j + W_j^1(\Delta - \rho_j) = \widehat{W}_j^0(\Delta, C_j)$ where the last equality is given by the fourth case of Eq. (9) when $c_j^* = C_j$. For $0 \leq \Delta < T_j$, it is also obvious that $\widehat{W}_j^0(\Delta, C_j) \geq \min\{\Delta, C_j\} = W_j^1(\Delta)$.

We then prove that $\widehat{W}_j^0(\Delta, C_j) \geq \widehat{W}_j^0(\Delta, c_j^*)$ for any $0 < c_j^* \leq C_j$ based on its definition in Eq. (9). Figure 5 provides an illustrative example for $\widehat{W}_j^0(\Delta, c_j^*)$. We consider three subcases:

- For $0 \leq \Delta \leq C_j$, it is obvious that $\widehat{W}_j^0(\Delta, C_j) \geq \widehat{W}_j^0(\Delta, c_j^*)$.
- For $C_j < \Delta \leq T_j - R_j + C_j$, we have $\widehat{W}_j^0(\Delta, C_j) = C_j$, and from Eq (9), $\widehat{W}_j^0(\Delta, c_j^*) = c_j^* + \max\{0, \Delta - (T_j - R_j + c_j^*)\} \leq c_j^* + C_j - c_j^* = C_j$.
- For $T_j - R_j + C_j < \Delta$, we have $\widehat{W}_j^0(\Delta, C_j) = C_j + W_j^1(\Delta - (T_j - R_j + C_j))$. Moreover, by definition, we also know $W_j^1(\Delta, c_j^*) \leq \delta + W_j^1(\Delta - \delta, c_j^*)$ for any δ such that $0 < \delta \leq \Delta$. Therefore, we conclude that $\widehat{W}_j^0(\Delta, c_j^*) = c_j^* + W_j^1(\Delta - (T_j - R_j + c_j^*)) \leq C_j + W_j^1(\Delta - (T_j - R_j + C_j))$ by setting δ to $C_j - c_j^*$ in the previous inequality.

□

Physical Meaning of Theorem 1

The rationale behind Theorem 1 may not be easy to be captured. A specific vector $\vec{x}$ defines how we plan to set the release jitter for each task as follows:

- For task τ_{k-1}, its release jitter is $R_{k-1} - C_{k-1}$ if x_{k-1} is 0 or S_{k-1} if x_{k-1} is 1.
- For task τ_j with $j = 1, 2, \ldots, k-2$, its release jitter is $Q_{j+1}^{\vec{x}} + R_j - C_j$ if x_j is 0 or $Q_{j+1}^{\vec{x}} + S_j$ if x_j is 1.

We use the following example to explain the physical meaning behind the setting of the release jitter of the tasks by referring to Step 3 in the proof of Theorem 1.

We consider Example 4 when ϵ is very close to 0. For such a case, $t_1^* = 4, t_2^* = 5, t_3^* = 5$, and $t_4^* = 6$. We consider $R_2 = 10$. By the above setting with $x_1 = 1, x_2 = 0, x_3 = 1$, we know that

- the release jitter of task τ_3 is 1 with the first release at time $t_4^* = 6$,
- the release jitter of task τ_2 is $1 + 10 - 1 = 10$ with the first release at time $t_4^* = 6$, and

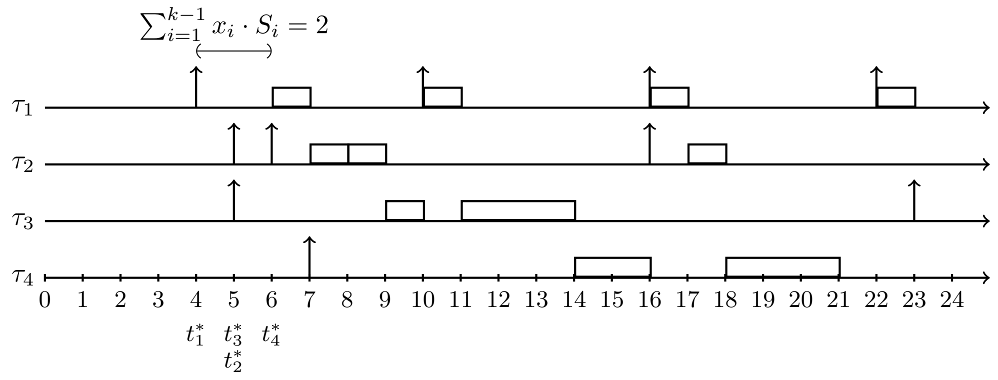


Fig. 6: An illustrative example for the physical meaning of Theorem 1 for Example 4.

- the release jitter of task τ_1 is $1+1=2$ with the first release at time $t_4^*=6$.

Or alternatively, we can equivalently rephrase it as follows:

- the release jitter of task τ_3 is 0 with the first release at time $t_3^*=5$,
- the release jitter of task τ_2 is $10-1=9$ with the first release at time $t_2^*=5$, and
- the release jitter of task τ_1 is 0 with the first release at time $t_1^*=4$.

Therefore, the response time analysis in Lemmas 13 and 14 can be explained as follows:

A safe scenario to analyze the worst-case response time R_k of task τ_k when $R_k \leq T_k$ is 1) to release each higher-priority task τ_j at time $t_j^* \overset{\text{def}}{=} \sum_{i=1}^{j-1} x_j S_j$ with release jitter $(1-x_j)(R_j - C_j)$, and 2) to execute the accumulated work only after time t_k^*, where t_1^* is an arbitrary constant.

Figure 6 provides a schedule based on the above setting. Note that self-suspension does not have to be accounted any more after the above transformation. Task τ_1 is an ordinary periodic task with period 6 with the first release at time 4, and task τ_3 is an ordinary periodic task with period 18 with the first release at time 5. Task τ_2 is a jittered periodic task with period 10 and 9 time-unit jitter, starting at time 5. Therefore, the second job of task τ_2 is released at time 6 in Figure 6.

The two idle time units are used between time 4 and time 6. These two time units are *blocked* simply for accounting the self-suspension behavior in $\mathbf{T}_1$, and no job is allowed to be executed in this time frame. The accumulated workload is then started to be executed at time 6 and the processor does not idle after time 6. Over here, we see that two jobs of task τ_2 are executed back to back from time 7 to time 9. As shown in Figure 6, the processor is busy executing the workload from time 6 to time 21. Therefore, we know that $21-6=15$ is a safe upper bound of R_4 in this example.

Proof of of Lemma 17. In this proof, we first transform the worst-case response time analysis presented in Corollary 1 in a more pessimistic analysis. We then prove that this more pessimistic version of Corollary 1 provides the same solution as Eq. (3), which then proves the lemma.

Since $Q_i^{\vec{x}} \overset{\text{def}}{=} \sum_{j=i}^{k-1} S_j \times x_j$, it holds that $Q_i^{\vec{x}} \leq Q_1^{\vec{x}}$ for $i = 1, 2, \ldots, k-1$. It follows that

$$\begin{aligned}
& C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_i^{\vec{x}} + (1-x_i)(R_i - C_i)}{T_i} \right\rceil C_i \\
\overset{(Q_i^{\vec{x}} \leq Q_1^{\vec{x}})}{\leq} & C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_1^{\vec{x}} + (1-x_i)(R_i - C_i)}{T_i} \right\rceil C_i \\
\overset{(R_i \leq D_i \leq T_i)}{\leq} & C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_1^{\vec{x}} + (1-x_i)T_i}{T_i} \right\rceil C_i \\
\overset{(x_i \in \{0,1\})}{=} & C_k + S_k + \sum_{i=1}^{k-1} (1-x_i) C_i + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_1^{\vec{x}}}{T_i} \right\rceil C_i
\end{aligned}$$

Therefore, the smallest positive value t such that

$$C_k + S_k + \sum_{i=1}^{k-1} (1-x_i) C_i + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_1^{\vec{x}}}{T_i} \right\rceil C_i \leq t \qquad (29)$$

is always larger than or equal to the solution of Eq. (5). Suppose that R_k^* is the smallest positive value t satisfying Eq. (29) for a given vector assignment $\vec{x}$.

Subtituting $(t + Q_1^{\vec{x}})$ by θ in Eq. (29), we get that R_k^* is upper bounded by the minimum value $(\theta - Q_1^{\vec{x}})$ greater than 0 (and therefore by the smallest $\theta > 0$) such that

$$\begin{aligned}
& C_k + S_k + \sum_{i=1}^{k-1} (1-x_i) C_i + \sum_{i=1}^{k-1} \left\lceil \frac{\theta}{T_i} \right\rceil C_i \leq \theta - Q_1^{\vec{x}} \\
\Leftrightarrow\ & C_k + S_k + Q_1^{\vec{x}} + \sum_{i=1}^{k-1} (1-x_i) C_i + \sum_{i=1}^{k-1} \left\lceil \frac{\theta}{T_i} \right\rceil C_i \leq \theta \\
\Leftrightarrow\ & C_k + S_k + \sum_{i=1}^{k-1} (x_i S_i + (1-x_i) C_i) + \sum_{i=1}^{k-1} \left\lceil \frac{\theta}{T_i} \right\rceil C_i \leq \theta.
\end{aligned} \qquad (30)$$

Suppose that $R_k^\dagger$ is the smallest positive value θ satisfying Eq. (30) for the given vector assignment $\vec{x}$. By definition, $R_k^\dagger = R_k^* + Q_1^{\vec{x}} \geq R_k^*$ for any given vector assignment $\vec{x}$. Now, consider the particular vector assignment $\vec{x}$ in which

$$x_i = \begin{cases} 1 & \text{if } S_i \leq C_i \\ 0 & \text{otherwise,} \end{cases}$$

for $i = 1, 2, \ldots, k-1$. By the definition of B_k (i.e., Section III-C), we get that

$$B_k = S_k + \sum_{i=1}^{k-1} \min(C_i, S_i) = S_k + \sum_{i=1}^{k-1} (x_i S_i + (1 - x_i) C_i)$$

Eq. (30) thus becomes identical to Eq. (3). Therefore, if Eq. (3) deems a task set as being schedulable, so does Corollary 1. $\square$