Response-Time Analysis for Self-Suspending Tasks Under EDF Scheduling

Federico Aromolo
Scuola Superiore Sant'Anna, Pisa, Italy

Alessandro Biondi
Scuola Superiore Sant'Anna, Pisa, Italy

Geoffrey Nelissen
Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract

The self-suspending task model proved to be particularly effective in capturing the timing behavior of real-time systems characterized by complex execution patterns, such as computation offloading to hardware accelerators, inter-core synchronization by means of multiprocessor locking protocols, and highly parallel computation. Most of the existing results for the timing analysis of self-suspending tasks do not support the widely adopted Earliest Deadline First (EDF) scheduling algorithm, being instead primarily focused on fixed-priority scheduling. This paper presents a response-time analysis for constrained-deadline self-suspending tasks scheduled under EDF on a uniprocessor system. The proposed analysis is based on a model transformation from self-suspending sporadic tasks to sporadic tasks with jitter, which can then be analyzed using a state-of-the-art analysis method for EDF scheduling. Experimental results are presented to compare the performance of the proposed technique in terms of schedulability ratio with that of the pessimistic suspension-oblivious approach and with a less general technique for task sets with implicit deadlines.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Software and its engineering → Real-time schedulability

Keywords and phrases Real-Time Systems, Schedulability Analysis, Self-Suspending Tasks, EDF Scheduling

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2022.13

Supplementary Material *Software (ECRTS 2022 Artifact Evaluation approved artifact)*: `https://doi.org/10.4230/DARTS.8.1.5`

1 Introduction

Modern embedded software systems are characterized by execution behaviors that are becoming increasingly complex. For instance, with the emergence of heterogeneous computing platforms that combine scalar multiprocessors with specialized hardware accelerators such as Field-Programmable Gate Arrays (FPGAs) and Graphics Processing Units (GPUs), the possibility of speeding up compute-intensive operations by offloading some computational activities to the accelerators has become commonplace. Complex execution behaviors also arise in multiprocessing due to inter-core synchronization. For example, this is the case for locking protocols that regulate the access to resources that are shared by tasks running on different processors, or for parallel tasks that dispatch computational activities upon multiple processors, with some of those activities being subject to precedence constraints specified according to a graph-based topology.

Such execution behaviors share the common pattern that some of the computational activities in the system may need to wait for some event to occur before continuing with their execution. In particular, for the case of computation offloading to hardware accelerators, the task performing the offloading must wait for the event signaling the completion of the accelerated workload. Instead, in the case of locking protocols, a task requesting access to a shared resource has to wait for the permission to access that resource in accordance to the specifics of the protocol. Finally, in the presence of precedence constraints, a subtask may have to wait for its predecessor subtasks to complete before starting its execution.

Since the delays incurred by a task when waiting for such events to occur may be significant, the typical implementation forces the task to relinquish the processor by having it suspend itself until the expected event occurs, as a way to avoid wasting processor time. The self-suspending task model was introduced to deal with the timing analysis of systems involving tasks that may suspend themselves to wait for an event to occur. This model has been extensively studied during the last decade, and proved to be a particularly effective tool to analyze the complex execution patterns exhibited by modern embedded systems from the point of view of their timing behavior [8].

Despite the effectiveness of EDF in dealing with both uniprocessor and multiprocessor scheduling problems, most of the existing analytical results for self-suspending tasks do not support the popular Earliest Deadline First (EDF) scheduling algorithm, being instead primarily focused on fixed-priority scheduling. In particular, none of the existing works provide a specialized and effective method to bound the response times of constrained-deadline self-suspending tasks in the specific case of uniprocessor systems.

Contributions. This paper presents a response-time analysis method for dynamic self-suspending tasks with constrained deadlines scheduled under EDF on a uniprocessor system. The analysis is based on a model transformation to the sporadic task model with release jitter and on the application of the exact worst-case response time (WCRT) analysis for sporadic tasks with jitter by Spuri [21]. An experimental comparison with the baseline suspension-oblivious approach, which pessimistically treats suspensions as additional computation [8], shows significant improvements in terms of the number of accepted task sets. For the less general case of task sets with implicit deadlines, in which the relative deadline of each task is equal to the minimum inter-arrival time of the task, the proposed approach is also compared with the state-of-the-art suspension-aware response time analysis by Günzel et al. [17]. In this case, the two approaches are shown to provide comparable performance.

Paper structure. The rest of this paper is organized as follows. Section 2 provides an overview of the literature on self-suspending task systems. Section 3 describes the system model and terminology considered in the paper. Section 4 presents the analytical derivation of the proposed approach and the resulting response-time analysis algorithm. The experimental results are reported and discussed in Section 5. Finally, Section 6 concludes the paper and discusses possible avenues for future work.

2 Related work

A comprehensive survey of the literature on self-suspending tasks was recently published by Chen et al. [8]. As discussed in that survey, many of the previous works on the analysis of real-time self-suspending tasks were found to be flawed. In addition to establishing a common framework for the analysis of self-suspending task systems, the survey by Chen et al. [8] aimed at collecting amendments to as many of those flawed works as possible.

Two main models exist for self-suspending tasks. The segmented self-suspending task model considers tasks whose execution behavior is determined by a fixed interleaving sequence of computation and suspension intervals, where each interval is characterized by a specific maximum length. The dynamic self-suspending task model, on the other hand, only assumes a total maximum execution time and a total maximum suspension time, computed, respectively, across all execution and suspension intervals. In an attempt to reduce the pessimism of response-time analyses for the dynamic self-suspending task model, von der Brüggen et al. [22] introduced the hybrid suspension model, which is similar to the dynamic model but assumes a limit on the maximum number of suspension intervals allowed for each job.

The typical analysis strategies for self-suspending tasks include modeling suspension time as computation, modeling the effect of suspension on other tasks as release jitter, and modeling the effect of suspension as a blocking term in the response-time analysis.

One of the most prominent works on the analysis of dynamic self-suspending tasks under uniprocessor fixed-priority scheduling is the work by Chen et al. [7], which proposed a response-time analysis for the dynamic self-suspending task model with constrained deadlines that dominates all other existing schedulability tests by combining elements of both the jitter-based and the blocking-based analyses. Similarly to the approach in the present paper, the proof for the analysis in [7] is based on a schedule transformation procedure followed by the analysis of the transformed schedule. Later, Günzel et al. [16] generalized the approach of [7] to the case where tasks have arbitrary deadlines and their releases are modeled by arrival curves.

For the case of segmented self-suspending tasks, Nelissen et al. [20] proposed a response-time analysis based on optimization methods for tasks with constrained deadlines scheduled under uniprocessor fixed-priority scheduling. For the case of multiprocessor systems, Liu and Anderson [18] derived the first suspension-aware WCRT analysis for dynamic self-suspending tasks under global scheduling. As discussed in [8], both the fixed-priority analyses from [20] and [18] required later revision due to some incorrect statements that were discovered within the respective proof frameworks.

Concerning the analysis of self-suspending task models under EDF scheduling, Liu and Anderson [18] also proposed a response-time analysis approach for multiprocessor global EDF scheduling of arbitrary-deadline tasks, which also supports soft real-time scheduling by means of tardiness thresholds. The approach by Dong and Liu [10] provides a utilization-based schedulability test for dynamic self-suspending tasks under multiprocessor global EDF scheduling for the case of implicit deadlines, and was later shown to be equivalent to the suspension-oblivious analysis for the case of uniprocessor systems [17]. Günzel et al. [17] provided the first response-time analysis for the dynamic model under EDF, for the case of implicit deadlines. That same work showed that an earlier analysis by Devi [9] that tried to solve the same problem was indeed flawed.

Self-suspending task models see fruitful application in the analysis of hardware-accelerated task systems in the context of heterogeneous computing. The case of hardware acceleration by means of Graphics Processing Units (GPUs) was explored in the works by Dong et al. [11] and Elliot et al. [12]. Biondi et al. [4] applied the segmented suspension model to the analysis of hardware acceleration on Field-Programmable Gate Arrays (FPGAs) embedded in emerging system-on-a-chip platforms.

Numerous works on the analysis of multiprocessor synchronization protocols hinge on self-suspending task models to derive a suitable real-time analysis. In this context, self-suspending task models can capture the behavior of tasks that suspend themselves while waiting to acquire a shared resource protected by a suspension-based locking mechanism. Detailed discussion on these works can be found in the most recent survey on multiprocessor locking protocols by Brandenburg [5].

Relevant applications of self-suspending task models also include the analysis of real-time parallel workloads. Fonseca et al. [14] considered a transformation to the segmented model for the analysis of parallel tasks under multiprocessor partitioned scheduling. The event-driven delay-induced (EDD) task model was introduced by Aromolo et al. [1] to model parallel topologies that incorporate delays in the concept of precedence constraints, with applications in the analysis of hardware-accelerated systems and of partitioned parallel tasks. The EDD task model was analyzed by means of a transformation to the dynamic self-suspending task model.

Concerning the analysis of non-suspending sporadic tasks with release jitter under uniprocessor EDF scheduling, the work by Spuri [21] generalizes previous results by Baruah et al. [2] to obtain both a feasibility test and an exact WCRT analysis based on the workload analysis approach. These approaches were later revised by George et al. [15] to provide some algorithmic efficiency improvements to the resulting analyses. To date, the technique by Spuri [21], combined with the efficiency enhancements in [15], represents a valid approach to check the feasibility of a set of sporadic tasks with jitter and to obtain their exact WCRTs.

3 System model

We consider a set $\Gamma = \{\tau_1, ..., \tau_n\}$ of n sporadic self-suspending real-time tasks executing on a single processor and described according to the dynamic self-suspending (DSS) task model [8]. Each task τ_i releases a potentially infinite sequence of jobs $\tau_{i,1}, \ldots, \tau_{i,j}, \ldots$ and is characterized by a tuple (C_i, S_i, T_i, D_i), where C_i represents the worst-case execution time (WCET), S_i is the maximum suspension time of each job of τ_i, T_i is the minimum inter-arrival time between the jobs of τ_i, and D_i is the relative deadline of the task, with $D_i \leq T_i$ (constrained deadlines). A job of τ_i may execute for up to C_i time units and may suspend itself at any point in its execution. When suspending, the job yields the processor for the execution of other tasks. The total suspension time of a job of τ_i across all its suspension intervals is upper bounded by S_i time units. The minimum inter-arrival time T_i represents the minimum amount of time separating successive jobs of τ_i.

We assume that job releases occur without jitter, so that each job $\tau_{i,j}$ is released as soon as it arrives. That is, if $a_{i,j}$ and $r_{i,j}$ represent, respectively, the arrival time and the release time of $\tau_{i,j}$, then it holds that $a_{i,j} = r_{i,j}$. Once it has arrived, a job $\tau_{i,j}$ is expected to complete its execution within D_i time units. Let $f_{i,j}$ denote the finishing time of a job $\tau_{i,j}$ of τ_i. We say that $\tau_{i,j}$ meets its deadline if $f_{i,j}$ is no greater than its absolute deadline $d_{i,j} = r_{i,j} + D_i$. The response time of a job $\tau_{i,j}$ is given by $R_{i,j} = f_{i,j} - a_{i,j} = f_{i,j} - r_{i,j}$. The worst-case response time (WCRT) R_i of a task τ_i is defined as the maximum possible response time across the jobs of τ_i. A job that is released and not yet completed is said to be pending.

Tasks in Γ are scheduled on the processor in a preemptive fashion according to the Earliest Deadline First (EDF) algorithm, which belongs to the class of job-level fixed-priority (JLFP) scheduling policies. Under EDF, each job $\tau_{i,j}$ is assigned a fixed priority level according to its absolute deadline $d_{i,j}$, such that a job with an earlier deadline has a higher priority. Ties are broken arbitrarily in case multiple jobs have the same absolute deadline.

4 Analysis

This section shows how to derive a schedulability test for a DSS task set Γ scheduled on a single processor under EDF scheduling, based on the response-time analysis (RTA) approach. In this approach, an upper bound $\overline{R}_i$ on the WCRT is derived for each task $\tau_i \in \Gamma$; then, the task set is deemed schedulable if $\overline{R}_i \leq D_i$ holds for every task $\tau_i \in \Gamma$.

The analysis for each task τ_i consists in deriving a transformation of the task set Γ to a task set Γ'_i of sequential sporadic real-time tasks with jitter, such that the WCRT R_i of τ_i in Γ is upper bounded by the WCRT R'_i of the corresponding task τ'_i in Γ'_i. The task set Γ'_i can then be analyzed according to the analysis by Spuri [21], hence obtaining a suitable upper bound for the response time of the DSS task τ_i.

For simplicity in the presentation, we assume that execution on the processor follows a discrete-time model where a unit of time corresponds to the length of the smallest relevant time scale in the system (e.g., the length of a processor cycle). The task schedule can then be seen as a sequence of time slices, each with a length of one time unit, within which the scheduling decisions are unaltered.

4.1 Sequential sporadic tasks with jitter

Since our proposed analysis is based on the idea of transforming the set of DSS tasks Γ into a set of sequential sporadic tasks with jitter, we introduce the terminology associated to sporadic tasks with jitter. Let $\Gamma' = \{\tau'_1, ..., \tau'_n\}$ represent a task set of sequential sporadic tasks with release jitter scheduled on a single processor under preemptive EDF. Each sporadic task with jitter τ'_i releases a potentially infinite sequence of jobs $\tau'_{i,1}, \ldots, \tau'_{i,j}, \ldots$ and is characterized by a tuple (C'_i, J'_i, T'_i, D'_i), where C'_i represents the worst-case execution time (WCET), J'_i is the maximum release jitter of each job of τ'_i, T'_i is the minimum inter-arrival time between the jobs of τ'_i, and D'_i is the relative deadline of the task, with $D'_i \leq T'_i$. The minimum inter-arrival time T'_i represents the minimum amount of time separating successive arrivals of jobs of τ'_i. The maximum release jitter J'_i is the maximum time a job of τ'_i can spend waiting for release after its arrival. Specifically, letting $a'_{i,j}$ and $r'_{i,j}$ represent, respectively, the arrival time and the release time of a job $\tau'_{i,j}$ of τ'_i, it holds that $r'_{i,j} - a'_{i,j} \leq J'_i$. Once it has arrived, a job $\tau'_{i,j}$ is expected to complete its execution within D'_i time units. Let $f'_{i,j}$ denote the finishing time of a job $\tau'_{i,j}$. The absolute deadline of $\tau'_{i,j}$ is defined as $d'_{i,j} = a'_{i,j} + D'_i$, and is considered respected if $f'_{i,j} \leq d'_{i,j}$. The response time of a job $\tau'_{i,j}$ is given by $R'_{i,j} = f'_{i,j} - a'_{i,j}$. The worst-case response time (WCRT) R'_i of a task τ'_i is defined as the maximum possible response time across the jobs of τ'_i.

4.2 Schedule transformation

By sustainability of self-suspending tasks with respect to their WCETs [6], the WCRT R_i of a task $\tau_i \in \Gamma$ is produced in a schedule σ of Γ in which all jobs $\tau_{j,l}$ of all tasks $\tau_j \in \Gamma$ execute up to their respective WCETs C_j. In the following procedure, we show how to transform the schedule σ in order to obtain a preemptive EDF schedule σ' in which none of the jobs self-suspend and where the response time of at least one of the jobs of τ'_i in σ' is equal to R_i.

Step 1 Initially set $\sigma' := \sigma$.

Step 2 Let $\tau_{i,k}$ represent a job of τ_i in σ with response time $R_{i,k} = R_i$. Let $\tau'_{i,k}$ represent the job of σ' corresponding to $\tau_{i,k}$. Remove all jobs in σ' with lower priority than $\tau'_{i,k}$, i.e., all jobs with deadline greater than $d'_{i,k}$.

Step 3 Replace all suspension intervals of jobs of τ'_i in σ' in which the processor is idle with execution intervals of equivalent length for τ'_i.

Step 4 Let t_f represent the finishing time of $\tau'_{i,k}$ and t_b represent the earliest time instant in σ' at and after which the processor is continuously busy until t_f, and let $I_B = [t_b, t_f)$ represent the busy interval for job $\tau'_{i,k}$. Identify the set of carry-in jobs C_B for the busy interval I_B as the set of jobs suspended at time $t_b - 1$ and that finish at or after t_b in σ'. Remove all jobs in σ' released before t_b that do not belong to C_B.

Step 5 Let t_0 represent the earliest release time among the jobs in C_B. Traverse all time slices in σ' within the interval $[t_0, t_b)$, from t_b down to t_0. For each such time slice T_I in which the processor is idle, if there is at least one time slice before T_I in which the processor is busy executing a job of C_B, let T_E represent the latest of such time slices and $\mathcal{J}_E$ represent the corresponding job, then, move the execution time of $\mathcal{J}_E$ in T_E from T_E to T_I.

Step 6 Let t'_b represent the (updated) earliest time instant in σ' at and after which the processor is continuously busy until t_f, and let $I'_B = [t'_b, t_f)$ represent the (extended) busy interval for job $\tau'_{i,k}$. For each job $\mathcal{J}_C$ in C_B, if $r'_C < t'_b$, where r'_C represents the release time of $\mathcal{J}_C$, postpone the release time r'_C of $\mathcal{J}_C$ to t'_b, without modifying the arrival time or the execution pattern of $\mathcal{J}_C$ in σ'. This corresponds to introducing a release jitter of length $t'_b - a'_C$ for $\mathcal{J}_C$, where a'_C represents the arrival time of $\mathcal{J}_C$.

Step 7 Remove all the execution that takes place at or after t_f from σ'.

Step 8 Traverse the processor time slices in σ' located within I'_B, from t'_b up to t_f. For each such time slice T_L, if a job $\mathcal{J}_L$ which is not one of the highest-priority jobs that are pending in T_L is executing in T_L, let $\mathcal{J}_H$ represent any one of the highest-priority jobs that are pending in T_L, and let T_H represent the earliest time slice after T_L in which the processor is busy executing $\mathcal{J}_H$, then, move the execution time of $\mathcal{J}_H$ in T_H from T_H to T_L and move the execution time of $\mathcal{J}_L$ in T_L from T_L to T_H.

4.2.1 Transformation example

Figure 1 provides an example that illustrates the schedule transformation procedure. In the provided schedules, upwards dashed arrows, upwards solid arrows, and downwards solid arrows represent, respectively, the arrival time, the release time, and the absolute deadline of a job, while white rectangles and grey rectangles represent, respectively, execution and self-suspension for a job.

Figure 1(a) illustrates an example schedule σ of a set of DSS tasks $\Gamma = \{\tau_1, \tau_2, \tau_3, \tau_4\}$. When applying the transformation procedure for task τ_1, the transformed schedule σ' is set to be identical to σ in **Step 1**. Assume that the job $\tau_{i,k}$ identified in **Step 2** corresponds to job $\tau_{1,3}$ in the example. Figure 1(b) shows the transformed schedule σ' after **Step 3** of the transformation. Then, the transformed schedule after **Step 6** is provided in Figure 1(c), where the busy interval $I_B = [t_b, t_f)$ and the extended busy interval $I'_B = [t'_b, t_f)$ are highlighted. The set of carry-in jobs C_B for I_B is composed of the first job of τ'_2 and the first job of τ'_3, and t_0 is set to coincide with the release of the first job of τ'_2. Note that, in **Step 6**, the release time of the first job of τ'_2 is delayed to coincide with t'_b. Finally, Figure 1(d) provides the resulting transformed schedule σ', obtained after **Step 8**. Note that the response time $R'_{1,3}$ of $\tau'_{1,3}$ was not altered in the transformation, i.e., $R'_{1,3} = R_{1,3}$.

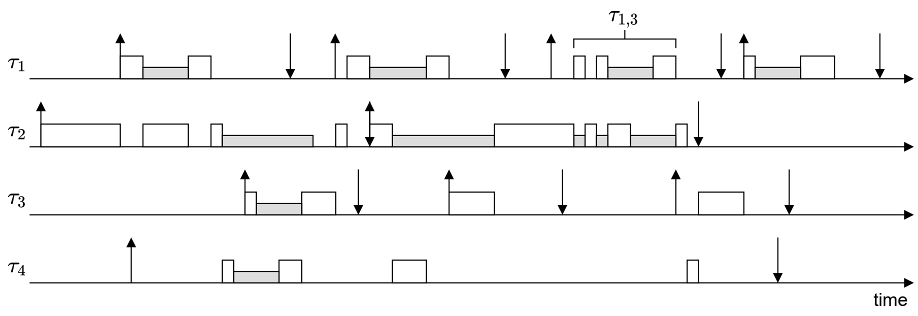
4.2.2 Properties of the transformed schedule

The following properties of σ' can be derived based on the transformation procedure.

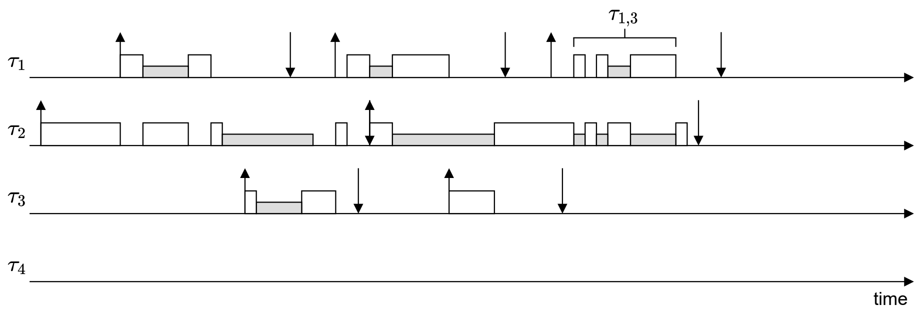
The following lemma establishes that the start of the extended busy interval t'_b happens at or before t_b.

▶ **Lemma 1.** *In the schedule σ', the extended busy interval I'_B starts at or before t_b; i.e., it holds that $t'_b \leq t_b$.*

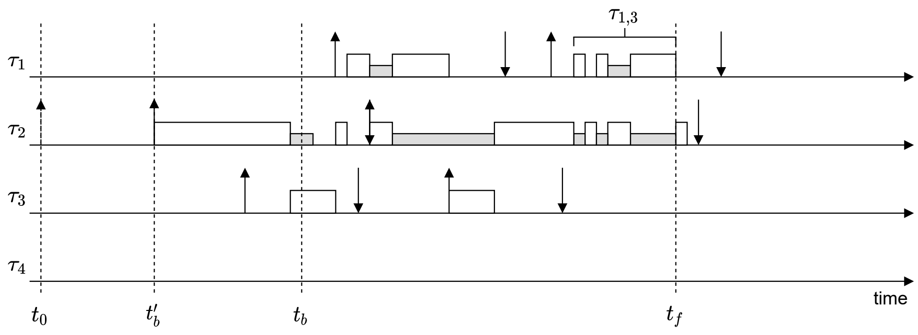
Proof. By definition, at the beginning of **Step 4**, the processor is continuously busy within the busy interval $I_B = [t_b, t_f)$, and, at the beginning of **Step 6**, the processor is continuously busy within the extended busy interval $I'_B = [t'_b, t_f)$. Note that the right end of the intervals



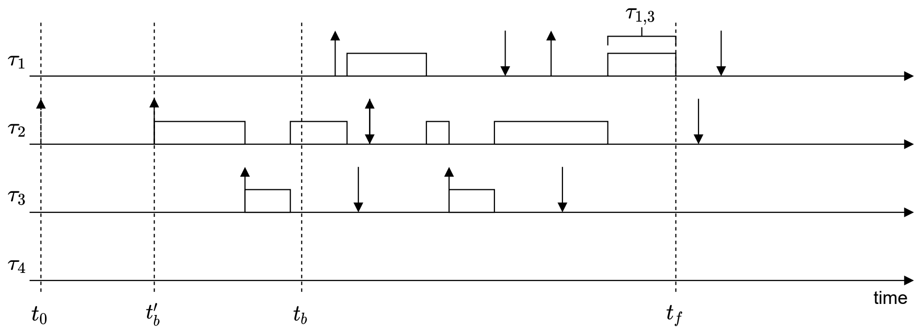
(a) Example schedule σ, equivalent to σ' after **Step 1** of the transformation.



(b) Example schedule σ' after **Step 3** of the transformation.



(c) Example schedule σ' after **Step 6** of the transformation.



(d) Example schedule σ' after **Step 8** of the transformation.

Figure 1 Transformation example.

I_B and I'_B is the same, since it is defined as t_f in both cases. Given these definitions, it is sufficient to show that I'_B cannot be smaller than I_B in order to prove the lemma. In other words, it must be shown that the interval where the processor is continuously busy until t_f at the beginning of **Step 6** cannot be shorter than the interval where the processor is continuously busy until t_f at the beginning of **Step 4**. In **Step 4**, jobs released before t_b and not belonging to C_B are removed from σ'. In order to derive a contradiction, consider one such job $\mathcal{J}_j$ and assume that it executed for at least one time slice within I_B before it was removed from σ' in **Step 4**. By the definition of the busy interval I_B, the processor must have been idle at time instant $t_b - 1$ at the beginning of **Step 4**. Therefore, $\mathcal{J}_j$ is suspended at $t_b - 1$. In fact, if $\mathcal{J}_j$ is not suspended at $t_b - 1$, then, by the work-conserving property of EDF, either $\mathcal{J}_j$ is executing at $t_b - 1$ or a job with higher or equal priority than $\mathcal{J}_j$ is executing at $t_b - 1$. This contradicts the fact that the processor is idle at $t_b - 1$. It follows that $\mathcal{J}_j$ satisfies the definition of carry-in job, since it is suspended at time $t_b - 1$ and it executes at or after t_b. This is in contradiction with the assumption that $\mathcal{J}_j$ does not belong to C_B. As a result, **Step 4** does not alter the execution pattern within $[t_b, t_f)$ in σ'. Finally, note that **Step 5** does not affect the execution pattern within $[t_b, t_f)$ in σ', since the execution slices from jobs in C_B can only be moved up to $t_b - 1$ in **Step 5**. Therefore, it holds that the busy interval I'_B identified at the beginning of **Step 6** can only be larger than or equal to I_B. ◀

The following lemma shows that the execution within σ' takes place wholly within the extended busy interval I'_B.

▶ **Lemma 2.** *In the schedule σ', the processor can be busy only within the extended busy interval I'_B.*

Proof. To obtain the lemma, we prove that (i) the processor cannot be busy at or after t_f and (ii) the processor cannot be busy before t'_b.

(i) In **Step 7**, all the execution that takes place at or after t_f is removed from σ'. Then, note that the execution slice interchanges within **Step 8** can only occur between time slices that were already busy at the end of **Step 7**. As a result, the processor cannot be busy at or after t_f in σ'.

(ii) In **Step 4**, all jobs released before t_b that do not belong to C_B are removed from σ'. Therefore, only jobs in C_B can execute before t_b in σ' after **Step 4**. In **Step 4**, t_0 is defined such that $t_0 \leq t_b$, and the execution slices of jobs in C_B are only moved within the interval $[t_0, t_b)$. Thus, no job of σ' executes before t_0 after **Step 5**. As a result, in case $t'_b = t_0$ in **Step 6**, no execution of jobs in C_B can take place before t'_b. Then, note that the case $t'_b < t_0$ cannot occur, since t'_b is defined in **Step 6** as the earliest time instant at and after which the processor is continuously busy until t_f, while the processor is idle before t_0 at the beginning of **Step 6**. In the following, consider the case in which $t'_b > t_0$. By definition of t'_b, the time slice $T_I = [t'_b - 1, t'_b)$ corresponding to $t'_b - 1$ is necessarily idle. Given that $t'_b > t_0$, and, by Lemma 1, $t'_b \leq t_b$, it holds that $T_I = [t'_b - 1, t'_b) \subseteq [t_0, t_b)$. If time slice T_I in $[t_0, t_b)$ is idle after **Step 5**, then it means there are no execution slices of jobs in C_B before T_I. Thus, no execution slices of jobs in C_B can take place before t'_b after **Step 6**. Finally, note that the execution slice interchanges within **Step 8** can only occur between time slices that were already busy after **Step 5**; therefore, no job of σ' executes before t'_b after **Step 8**. ◀

The following lemma shows that the processor is continuously busy within the extended busy interval I'_B after the transformation procedure.

▶ **Lemma 3.** *In the schedule σ', the processor is continuously busy within the extended busy interval I'_B.*

Proof. In **Step 6**, the extended busy interval $I'_B = [t'_b, t_f)$ is identified by construction as an interval in which the processor is continuously busy. Removing the execution of jobs in σ' that takes place at or after t_f in **Step 7** does not alter the execution pattern within I'_B. Then, the execution slice interchanges within **Step 8** can only occur between time slices that were already busy at the beginning of **Step 6**. Therefore, the processor is continuously busy within the extended busy interval I'_B in the transformed schedule σ'. ◀

The following lemma shows that jobs of the task under analysis τ'_i cannot be categorized as carry-in jobs in **Step 4** of the schedule transformation.

▶ **Lemma 4.** *The set C_B of carry-in jobs for the busy interval I_B does not contain any job of τ'_i.*

Proof. By the definition of the busy interval I_B in **Step 4**, the processor is idle at time instant $t_b - 1$. Then, in order for a job $\tau'_{i,l}$ of τ'_i to belong to the set C_B, it must hold that $\tau'_{i,l}$ is suspended when the processor is idle at time $t_b - 1$ in **Step 4**. This is impossible since, in **Step 3**, all suspension intervals of jobs of τ'_i in which the processor is idle are replaced with execution intervals of equivalent length for that job. ◀

The following lemma shows that the response time of job $\tau_{i,k}$ is preserved after the transformation.

▶ **Lemma 5.** *The response time of job $\tau'_{i,k}$ in σ' is equal to the response time of $\tau_{i,k}$ in σ, i.e., it holds that $R'_{i,k} = R_{i,k}$.*

Proof. In **Step 2**, all jobs with priority lower than that of $\tau'_{i,k}$ are removed from the schedule σ'. Therefore, $\tau'_{i,k}$ is one of the jobs that share the lowest priority in the schedule σ'. In **Step 3**, additional execution slices of $\tau'_{i,k}$ can only be added before its finishing time $f'_{i,k} = f_{i,k}$. Then, in **Step 4**, only jobs that are released before t_b can be removed from σ'. Note that, by definition, the right end of the busy interval $I_B = [t_b, t_f)$ defined in **Step 4** corresponds to the finishing time $f'_{i,k}$ of $\tau'_{i,k}$, with $f'_{i,k} = f_{i,k}$. In addition, in **Step 3**, all suspension intervals of $\tau'_{i,k}$ in which the processor is idle are replaced with execution intervals of equivalent length for $\tau'_{i,k}$; thus, within the interval $[a'_{i,k}, f'_{i,k})$, the processor is either busy executing $\tau'_{i,k}$ or another job with higher or equal priority than $\tau'_{i,k}$. It follows that the start of the busy period t_b must necessarily occur at or before the arrival time $a'_{i,k}$ of $\tau'_{i,k}$. Therefore, **Step 4** does not affect the execution of $\tau'_{i,k}$. Similarly, **Step 5** only affects the execution of jobs belonging to C_B, which, by definition, are released before t_b. Thus, the execution of $\tau'_{i,k}$ is not affected by **Step 5**. Then, in **Step 7**, removing the execution of jobs in σ' taking place at or after t_f does not affect the finishing time of $\tau'_{i,k}$. Finally, in **Step 8**, execution slices of a job $\mathcal{J}_H$ can only be anticipated to an earlier time slice T_E if $\mathcal{J}_H$ has higher priority than the job executing in T_E. Since $\tau'_{i,k}$ is one of the jobs that share the lowest priority in the schedule σ', it is not possible that the final execution slice of $\tau'_{i,k}$ is anticipated in **Step 8**. Therefore, the finishing time of $\tau'_{i,k}$ after the transformation is given by $f'_{i,k} = f_{i,k}$. In addition, since arrival times for jobs of σ' are not modified with respect to the corresponding jobs of σ within the transformation procedure, it holds that $a'_{i,k} = a_{i,k}$. It follows that $R'_{i,k} = f'_{i,k} - a'_{i,k} = f_{i,k} - a_{i,k} = R_{i,k}$. ◀

The following lemma shows that jobs in σ' are scheduled in accordance with the preemptive EDF scheduling policy.

▶ **Lemma 6.** *For any time slice T_σ in the schedule σ', the processor executes one of the jobs with earliest absolute deadline that are pending in T_σ, if any.*

Proof. The statement trivially holds for any time slice of σ' in which no job is ready for execution. By Lemma 2, the time slices in which the processor is busy are limited to the busy interval I'_B. Concerning the time slices within I'_B, in **Step 8**, for each time slice T_L within I'_B, the schedule σ' is modified such that one of the highest-priority jobs that are pending in T_L is executing in T_L. The statement follows since the priority ordering in **Step 8** is determined according to the EDF scheduling policy. ◀

The following lemma guarantees that the execution of a job in σ' cannot occur before the release time of the corresponding job in σ.

▶ **Lemma 7.** *Consider a job $\tau'_{j,l}$ of a task τ'_j in σ'. The execution of $\tau'_{j,l}$ that takes place in σ' does not start before $r_{j,l}$.*

Proof. The execution of the job $\tau_{j,l}$ corresponding to job $\tau'_{j,l}$ in σ cannot start before its release time $r_{j,l}$. In the following, we show that no execution slices for job $\tau'_{j,l}$ are added before $r_{j,l}$ within the transformation procedure. In **Step 3**, additional execution time can only be introduced for a job $\tau'_{j,l}$ in a time slice T_S where $\tau'_{j,l}$ is suspended in σ, meaning that $\tau'_{j,l}$ was released before T_S. In **Step 5**, execution slices of jobs in C_B can only be delayed to a later time slice, meaning that no execution slices of $\tau'_{j,l}$ are introduced before $r_{j,l}$. Finally, consider that, in case an execution slice of $\tau'_{j,l}$ is moved as part of **Step 8**, then either one of the following scenarios occurs:

1. An execution slice of $\tau'_{j,l}$ originally occurring at T_H is anticipated to T_L. By construction of **Step 8**, in order for this situation to occur, $\tau'_{j,l}$ must have been pending in T_L, meaning that it was released at or before the start of T_L.
2. An execution slice of $\tau'_{j,l}$ originally occurring at T_L is delayed to T_H, which occurs after T_L by construction. Since job $\tau'_{j,l}$ was executing in T_L, the release time $r_{j,l}$ must have occurred at or before the start of T_H.

As a result, no matter how many times the execution slices of $\tau'_{j,l}$ are moved in **Step 8**, no execution slices of $\tau'_{j,l}$ are introduced before $r_{j,l}$. ◀

The following lemma shows that jobs in σ' do not execute before their release time.

▶ **Lemma 8.** *Consider a job $\tau'_{j,l}$ of a task τ'_j in σ'. Job $\tau'_{j,l}$ does not execute before its release time $r'_{j,l}$.*

Proof. The release time of a job $\tau'_{j,l}$ in σ' can only be modified in **Step 6** of the transformation. In case $\tau'_{j,l}$ does not belong to C_B, its release time is not modified in **Step 6**, meaning that $r'_{j,l} = r_{j,l}$ in the transformed schedule σ'. By Lemma 7, the execution of a job $\tau'_{j,l}$ in σ' does not start before $r_{j,l}$. Therefore, in case $\tau'_{j,l} \notin C_B$, $\tau'_{j,l}$ does not execute before $r'_{j,l}$ in σ'. Similarly, in case $\tau'_{j,l} \in C_B$ and $r'_{j,l} \geq t'_b$ at the beginning of **Step 6**, the release time of $\tau'_{j,l}$ is not modified, therefore $r'_{j,l} = r_{j,l}$, and, by Lemma 7, the execution of $\tau'_{j,l}$ in σ' does not start before $r_{j,l}$. Finally, in case $\tau'_{j,l} \in C_B$ and $r'_{j,l} < t'_b$ at the beginning of **Step 6**, then the release time of $\tau'_{j,l}$ is set to $r'_{j,l} = t'_b$. By Lemma 2, none of the jobs in σ' execute outside the busy interval $I'_B = [t'_b, t_f)$, thus $\tau'_{j,l}$ does not execute before $r'_{j,l} = t'_b$. ◀

The following lemma provides an upper bound on the release jitter introduced for the jobs in σ'.

▶ **Lemma 9.** *Consider a job $\tau'_{j,l}$ of a task τ'_j in σ'. The release jitter of $\tau'_{j,l}$ is upper bounded by $R_j - C_j$.*

Proof. Since arrival times are not altered for jobs of σ' with respect to σ, it holds that $a'_{j,l} = a_{j,l}$. Within the schedule transformation, the release time of jobs in σ' can only be postponed for jobs of C_B (in **Step 6**). Therefore, in case $\tau'_{j,l}$ does not belong to C_B, the release jitter of $\tau'_{j,l}$ is given by $r'_{j,l} - a'_{j,l} = r_{j,l} - a_{j,l} = 0$. In the following, consider the case in which $\tau'_{j,l}$ belongs to C_B. In case the release time $r'_{j,l}$ is not modified in **Step 6**, then the release jitter of $\tau'_{j,l}$ is 0. Otherwise, the release time $r'_{j,l}$ is set to t'_b. Therefore, in the latter case, the resulting release jitter for $\tau'_{j,l}$ is given by $r'_{j,l} - a'_{j,l} = t'_b - a_{j,l}$. By definition of carry-in job, the finishing time of $\tau'_{j,l}$ is greater than or equal to t_b in **Step 4**. Furthermore, because **Step 5** does not alter the schedule σ' at or after t_b, the finishing time of $\tau'_{j,l}$ is unchanged at the end of **Step 5**, i.e., just before the release time $r'_{j,l}$ is postponed in **Step 6** to yield the release jitter of $\tau'_{j,l}$. In addition, note that, by the assumption that job $\tau_{j,l}$ executes for its WCET C_j in σ, and since none of the steps in the transformation up to **Step 6** reduce or increase the total execution time of any job in C_B, $\tau'_{j,l}$ executes for C_j units of time in σ' at the beginning of **Step 6**. Finally, by Lemma 8, the execution of $\tau'_{j,l}$ only takes place at or after $r'_{j,l} = t'_b$ in **Step 6**. It follows that, at the beginning of **Step 6**, $f_{j,l} \geq t'_b + C_j$. As a result, the release jitter introduced for $\tau'_{j,l}$ in **Step 6** can be upper bounded as $t'_b - a_{j,l} \leq f_{j,l} - C_j - a_{j,l} = R_{j,l} - C_j \leq R_j - C_j$. ◀

4.3 Model transformation to enable the response-time analysis

To prove that the schedule σ' resulting from the schedule transformation is suitable to be analyzed as a set of sequential sporadic tasks with jitter, we first define a legal schedule for a set of sequential sporadic tasks with jitter under preemptive EDF scheduling.

▶ **Definition 10.** *A schedule σ' is considered legal with respect to preemptive EDF scheduling of a set Γ' of sequential sporadic tasks with jitter if the following statements hold for each job $\tau'_{j,l}$ of all tasks τ'_j in σ':*

- ***Property 1.*** *The minimum inter-arrival time constraint is satisfied; i.e., if $l > 1$, it holds that $a'_{j,l} \geq a'_{j,l-1} + T'_j$.*
- ***Property 2.*** *The absolute deadline $d'_{j,l}$ of $\tau'_{j,l}$ is such that $d'_{j,l} = a'_{j,l} + D'_j$.*
- ***Property 3.*** *The processor does not execute $\tau'_{j,l}$ for more than C'_j units of time.*
- ***Property 4.*** *The release of $\tau'_{j,l}$ takes place at or after its arrival; i.e., it holds that $r'_{j,l} \geq a'_{j,l}$.*
- ***Property 5.*** *The processor does not execute $\tau'_{j,l}$ before its release time $r'_{j,l}$.*
- ***Property 6.*** *The release jitter constraint is satisfied, i.e., it holds that $r'_{j,l} - a'_{j,l} \leq J'_j$.*
- ***Property 7.*** *For each time slice from the release time $r'_{j,l}$ up to the finishing time $f'_{j,l}$ of $\tau'_{j,l}$, the processor is either busy executing $\tau'_{j,l}$ or another job with absolute deadline smaller or equal than that of $\tau'_{j,l}$.*

The following lemma shows that σ' is a legal preemptive EDF schedule of a set of sequential sporadic tasks with jitter.

▶ **Lemma 11.** *Consider a task set $\Gamma'_i = \{\tau'_1, ..., \tau'_n\}$ of sequential sporadic tasks with release jitter, with $\tau'_i = (C_i + S_i, 0, T_i, D_i)$ and $\tau'_j = (C_j, R_j - C_j, T_j, D_j)$ for all $\tau'_j \neq \tau'_i$. The transformed schedule σ' is a legal schedule of Γ'_i under preemptive EDF scheduling.*

Proof. In the following, we show that σ' complies with Definition 10 with respect to task set Γ'_i. First, note that the arrival times and the absolute deadlines of the jobs in σ' are kept equal to the arrival times and absolute deadlines of the corresponding jobs in σ. Therefore, Property 1 and Property 2 in Definition 10 hold for each job in σ'. Then, consider that the schedule transformation procedure does not increment the cumulative execution time of jobs of τ'_j in σ' such that $\tau'_j \neq \tau'_i$, thus the amount of execution for such jobs is within C_j. In addition, the execution time of jobs of τ'_i is not incremented by more than S_i (in **Step 3**). Therefore, Property 3 holds for each job in σ'. Within the schedule transformation, the release times of jobs in σ' can only be modified for jobs of C_B (in **Step 6**). By Lemma 4, jobs of τ'_i do not belong to the set C_B; therefore, it holds that $r'_{i,l} = r_{i,l}$ for each job $\tau'_{i,l}$ of τ'_i. As a result, $r'_{i,l} = a'_{i,l}$ holds for each job $\tau'_{i,l}$ of τ'_i. Then, consider a job $\tau'_{j,l}$ that belongs to C_B. If at the beginning of **Step 6** $r'_{j,l} < t'_b$, then the release time of $\tau'_{j,l}$ is postponed to t'_b; therefore, it holds that $r'_{j,l} \geq a'_{j,l}$ in the transformed schedule σ'. Otherwise, the release time of $\tau'_{j,l}$ is not modified, i.e., $r'_{j,l} = a'_{j,l}$. Thus, Property 4 holds for all jobs in σ'. Property 5 and Property 6 follow directly from Lemma 8 and Lemma 9, respectively, for each job in σ'. Finally, by Lemma 6, for any time slice T_σ in the schedule σ', the processor executes one of the jobs with earliest absolute deadline that are pending in T_σ, if any. Therefore, Property 7 holds for each job in σ'. ◀

The following theorem shows how to obtain a task set Γ'_i of sequential sporadic tasks with jitter that can be analyzed in order to obtain a WCRT upper bound for task τ_i.

▶ **Theorem 12.** *Given a task set Γ of DSS tasks, the WCRT R_i of a task $\tau_i \in \Gamma$ is upper bounded by the WCRT R'_i of a sequential sporadic task τ'_i in Γ'_i, where $\Gamma'_i = \{\tau'_1, ..., \tau'_n\}$ is a set of sequential sporadic tasks with release jitter, with $\tau'_i = (C_i + S_i, 0, T_i, D_i)$ and $\tau'_j = (C_j, R_j - C_j, T_j, D_j)$ for all $\tau'_j \neq \tau'_i$.*

Proof. By Lemma 5, the response time of job $\tau'_{i,k}$ in σ' is equal to the response time of $\tau_{i,k}$ in σ, which is in turn equivalent to the WCRT R_i of the task under analysis τ_i; i.e., it holds that $R'_{i,k} = R_{i,k} = R_i$. In addition, since by Lemma 11 σ' represents a legal schedule of Γ'_i under preemptive EDF scheduling, it holds that $R'_{i,k} \leq R'_i$. It follows that $R_i \leq R'_i$. ◀

4.4 Response-time analysis algorithm

The schedulability of a task set Γ of DSS tasks can be verified by means of the iterative approach described in Algorithm 1. The algorithm produces a vector of WCRT upper bounds $\overline{\mathbf{R}} = [\overline{R}_1, ..., \overline{R}_n]$ for all tasks in Γ using Theorem 12 (ModelTransformation at line 10), starting from the initial condition in which $\overline{R}_i$ is set to D_i for each $\tau_i \in \Gamma$. Then, the algorithm iteratively applies Theorem 12 to each task $\tau_i \in \Gamma$ in order to obtain the WCRT R'_i of task τ'_i by means of the response-time analysis approach by Spuri [21] (JitterAnalysis at line 11). At each iteration, the value of $\overline{R}_i$ in $\overline{\mathbf{R}}$ is set to the newly obtained R'_i in case $R'_i < \overline{R}_i$, and the task set is deemed schedulable if $R'_i \leq D_i$ holds for each $\tau_i \in \Gamma$. Otherwise, the iterative loop continues until either the task set is deemed schedulable or the vector $\overline{\mathbf{R}}$ is not updated within the iteration, in which case the task set is deemed not schedulable.

The use of this iterative algorithm is supported by the following reasoning. Assume that the behavior of the preemptive EDF scheduler is altered by means of a run-time mechanism $\mathcal{M}$ that forcibly terminates all jobs at the time of their absolute deadline. With mechanism $\mathcal{M}$ in place, the relative deadline D_i represents a valid upper bound on the WCRT R_i of a task τ_i in Γ. Then, if in a given iteration of the algorithm the WCRT upper bound R'_i obtained for each task τ_i is found to be lower than or equal to the corresponding deadline D_i, then all the possible jobs of all tasks in Γ will terminate within their absolute deadline. In this situation, the mechanism $\mathcal{M}$ does not need to prevent execution for any job, therefore its presence is irrelevant and the resulting behavior is equivalent to standard preemptive

EDF scheduling, wherein $\mathcal{M}$ is not deployed. Otherwise, if in the same iteration at least one of the WCRT upper bounds R'_i for a task τ_i is found to be greater than the corresponding relative deadline D_i, then the task set cannot be deemed schedulable at that iteration. Then, if at least one of the WCRT upper bounds in $\overline{\mathbf{R}}$ was updated, the algorithm proceeds to the next iteration to potentially reduce the WCRT upper bounds of the other tasks in Γ. This reasoning was also adopted in [19] to obtain a similar iterative approach for the derivation of WCRT upper bounds in systems where tasks synchronize their access to shared resources. Note that, by construction, in a given iteration beyond the first, each of the values in $\overline{\mathbf{R}}$ is less or equal than the corresponding value at the previous iteration, and that the algorithm terminates as soon as none of the values in $\overline{\mathbf{R}}$ are updated after an iteration.

When applying Theorem 12 within the algorithm to analyze a task $\tau_i \in \Gamma$ (at line 10), note that the exact value of the WCRT R_j of each task $\tau_j \neq \tau_i$ is not known; therefore, when constructing Γ'_i, $\overline{R}_j - C_j$ must be used as an upper bound of the jitter of τ'_j in place of $J'_j = R_j - C_j$. To ensure that the algorithm remains consistent with this substitution, consider a generic iteration of the algorithm. In case $\overline{R}_j$ was never updated during the previous iterations of the algorithm, then $\overline{R}_j = D_j$, and, assuming $\mathcal{M}$ is active, $D_j = \overline{R}_j \geq R_j$. Instead, if $\overline{R}_j$ was updated in at least one of the previous iterations of the algorithm, then $\overline{R}_j$ must have been set to $\overline{R}_j = R'_j$, where R'_j was obtained by analyzing τ_j by means of task set Γ'_j in Theorem 12 with respect to previous values of the WCRT upper bounds in $\overline{\mathbf{R}}$. In this case, by Theorem 12, it holds that $R_j \leq R'_j$, with R'_j computed with respect to Γ'_j; thus, $R_j \leq \overline{R}_j$. As a result, $\overline{R}_j \geq R_j$ holds for both cases. Therefore, since $\overline{R}_j - C_j \geq R_j - C_j$, and since increasing the maximum jitter parameter for a task in a task set of sporadic tasks with jitter cannot reduce the WCRT of tasks in that task set, $\overline{R}_j - C_j$ can be used as a safe upper bound on the jitter J'_j of τ'_j for the analysis of τ_i.

Note that the response-time analysis by Spuri [21] can only be applied to systems that are not overloaded, i.e., to those systems for which the system utilization factor $U' = \sum_{\tau'_i \in \Gamma'} \frac{C'_i}{T'_i}$ does not exceed one. Therefore, this condition must be verified in Algorithm 1 before a task set Γ'_i can be analyzed. Given a task set Γ'_i generated to analyze a task τ_i in Γ, this precondition on the system utilization is satisfied if $\frac{S_i}{T_i} + \sum_{\tau_j \in \Gamma} \frac{C_j}{T_j} \leq 1$. This is because the WCET C'_i of τ'_i is incremented by S_i with respect to the original task $\tau_i \in \Gamma$, while the WCET C'_j of tasks $\tau'_j \neq \tau'_i$ is kept equal to C_j. However, note that the resulting utilization factor for each task set Γ'_i to be analyzed in Algorithm 1 is independent of the values of the jitter J'_i, i.e., it is independent of the values of the elements of $\overline{\mathbf{R}}$. Therefore, it is sufficient to check if $\frac{S_i}{T_i} + \sum_{\tau_j \in \Gamma} \frac{C_j}{T_j} \leq 1$ holds for each task under analysis $\tau_i \in \Gamma$ before starting the iterative refinement of the WCRT upper bounds $\overline{\mathbf{R}}$ in Algorithm 1 (NecessaryConditions at line 2).

Finally, note that the iterative loop does not need to terminate immediately in case the task set is deemed to be schedulable (i.e., when $R'_i \leq D_i$ holds for each $\tau_i \in \Gamma$). In fact, it is possible to obtain tighter WCRT upper bounds by performing additional iterations with the updated vector $\overline{\mathbf{R}}$.

5 Experimental results

This section presents the results of an experimental evaluation of the proposed response-time analysis approach. For the case of constrained deadlines, we propose a comparison with the suspension-oblivious approach. Then, for the case of implicit deadlines, where the relative deadline of each task is equal to its minimum inter-arrival time, the proposed approach is also compared with the response-time analysis technique by Günzel et al. [17].

Algorithm 1 Schedulability analysis for a task set Γ.

```
1: procedure SchedulabilityTest(Γ)
2:     if NecessaryConditions(Γ) = FALSE then
3:         return FALSE
4:     end if
5:     $\forall \tau_i \in \Gamma, \overline{R}_i \leftarrow D_i$
6:     update ← TRUE
7:     while update = TRUE do
8:         update ← FALSE
9:         for all $\tau_i \in \Gamma$ do
10:            $\Gamma'_i \leftarrow$ ModelTransformation($\Gamma, i, \overline{\mathbf{R}}$)
11:            $R'_i \leftarrow$ JitterAnalysis($\Gamma'_i, i$)
12:            if $R'_i < \overline{R}_i$ then
13:                $\overline{R}_i \leftarrow R'_i$
14:                update ← TRUE
15:            end if
16:        end for
17:        if $\forall \tau_i \in \Gamma, R'_i \leq D_i$ then
18:            return TRUE
19:        end if
20:    end while
21:    return FALSE
22: end procedure
```

5.1 Experimental setup

The proposed experiments are based on the analysis of randomly generated task sets. The task set generator used in the experiments was instrumented as follows. The number of tasks generated for each task set is set to a fixed number n. For each task set, the UUniFast algorithm by Bini and Buttazzo [3] was used to generate a set of utilization values U_i, such that $U = \sum_{\tau_i \in \tau_i} U_i$, where U is a generation parameter representing the system utilization, which is varied within the experiments. For each task τ_i in the randomly generated task set, the minimum inter-arrival time was selected from a discrete log-uniform distribution in the range $[T_{min}, T_{max}]$, where T_{min} and T_{max} are generation parameters representing the minimum and the maximum possible value of T_i, as suggested by Emberson et al. [13]. The WCET of τ_i was then set to $C_i = U_i \cdot T_i$. The maximum suspension time S_i of τ_i was selected from a discrete uniform distribution in the range $[(T_i - C_i) \cdot \beta_{min}, (T_i - C_i) \cdot \beta_{max}]$, where β_{min} and β_{max} are generation parameters such that $\beta_{min} \in [0, 1]$ and $\beta_{max} \in [0, 1]$. The relative deadline D_i of τ_i was selected from a discrete uniform distribution in the range $[C_i + (T_i - C_i) \cdot \alpha, T_i]$, where α is a generation parameter such that $\alpha \in [0, 1]$ for the experiments with constrained deadlines, so that $D_i \leq T_i$, and is instead equal to 1 for the experiments with implicit deadlines, so that $D_i = T_i$. In the experiments, the system utilization U is varied from 0 to 1 in increments of 0.05. For each value of U, 1000 task sets were tested using the approaches under evaluation. The performance metric for the experiments is the schedulability ratio with respect to a specific system utilization U; i.e., the ratio of task sets deemed schedulable by a specific analysis approach over the number of task sets generated for the system utilization point U.

5.2 Results with constrained deadlines

For the case of constrained deadlines, the performance of the proposed approach (**SS-RTA**) in terms of schedulability ratio is compared with that of the suspension-oblivious RTA approach (**SO-RTA**). In the **SO-RTA** approach, suspensions are regarded as additional computation time, and the resulting task set is evaluated with the EDF analysis by Spuri [21]. In particular, the task set analyzed in **SO-RTA** is constructed as $\Gamma' = \{\tau_1', \ldots, \tau_n'\}$, where $\tau_i' = (C_i + S_i, 0, T_i, D_i)$ for each $\tau_i \in \Gamma$. Figures 2(a)-(c) report the results of the experiments with constrained deadlines. In these experiments, the values of T_{min} and T_{max} were set to 100 and 1000, respectively. Figure 2(a) shows that the proposed approach outperforms the suspension-oblivious approach by a significant margin, even with moderate amounts of suspension. When the parameter β_{max} is increased to 0.6 (Figure 2(b)), the schedulability ratio obtained with the suspension-oblivious analysis approaches 0, even for low values of utilization. As shown in Figure 2(c), the proposed approach retains significant schedulability ratios even with the shorter deadlines introduced by generating task sets with shorter relative deadlines (i.e., with a smaller value of α).

5.3 Results with implicit deadlines

The results of the experiments on implicit deadlines are provided in Figures 2(d)-(f). In these experiments, the proposed approach (**SS-RTA**) is compared with the suspension-oblivious approach (**SO-RTA**) and the state-of-the-art RTA for implicit deadlines by Günzel et al. [17] (**SA-RTA**). In this case, for the **SO-RTA** approach, it is sufficient to inflate the WCETs of each task by the maximum suspension time and to check whether the utilization of the resulting task set is less than or equal to 1. In addition, the performance of the schedulability test obtained with the logic OR of **SS-RTA** and **SA-RTA**, which deems a task set under analysis schedulable in case at least one of **SS-RTA** and **SA-RTA** deems the task set schedulable, is reported in the experiments (**OR-RTA**). The values of T_{min} and T_{max} are again set to 100 and 1000, respectively. Figure 2(d) shows that, when relatively small values of β_{min} and β_{max} are applied, **SA-RTA** outperforms **SS-RTA** by a slight margin. Nonetheless, it should be noted that the combination of the two approaches (**OR-RTA**) provides some improvement over using **SA-RTA** by itself. This means that the two approaches are not comparable, in the sense that there exist task sets that are deemed schedulable by **SS-RTA** and that are deemed not schedulable by **SA-RTA**, and vice-versa. The gap between the two approaches is reduced with larger values for the maximum suspension time of the generated tasks, i.e., when β_{max} is increased to 0.6 (Figure 2(e)). Finally, Figure 2(f) shows that the the performance of the proposed approach surpasses the performance of **SA-RTA** when more tasks are included in each task set ($n = 15$).

Overall, these experiments show that the performance levels of the proposed approach **SS-RTA** and of the state-of-the art approach **SA-RTA** are comparable, and that neither of the methods dominates the other. In fact, the strongest performance is obtained with the combined approach **OR-RTA**, which theoretically dominates both approaches and provides slight empirical improvements under specific system configurations. It should be noted that the main advantage of **SS-RTA** over **SA-RTA** is that it allows evaluating task sets with constrained deadlines in addition to task sets with implicit deadlines. Finally, the experiments show that both approaches vastly outperform the basic suspension-oblivious approach, which is only capable of accepting a very limited number of task sets under the evaluated scenarios.

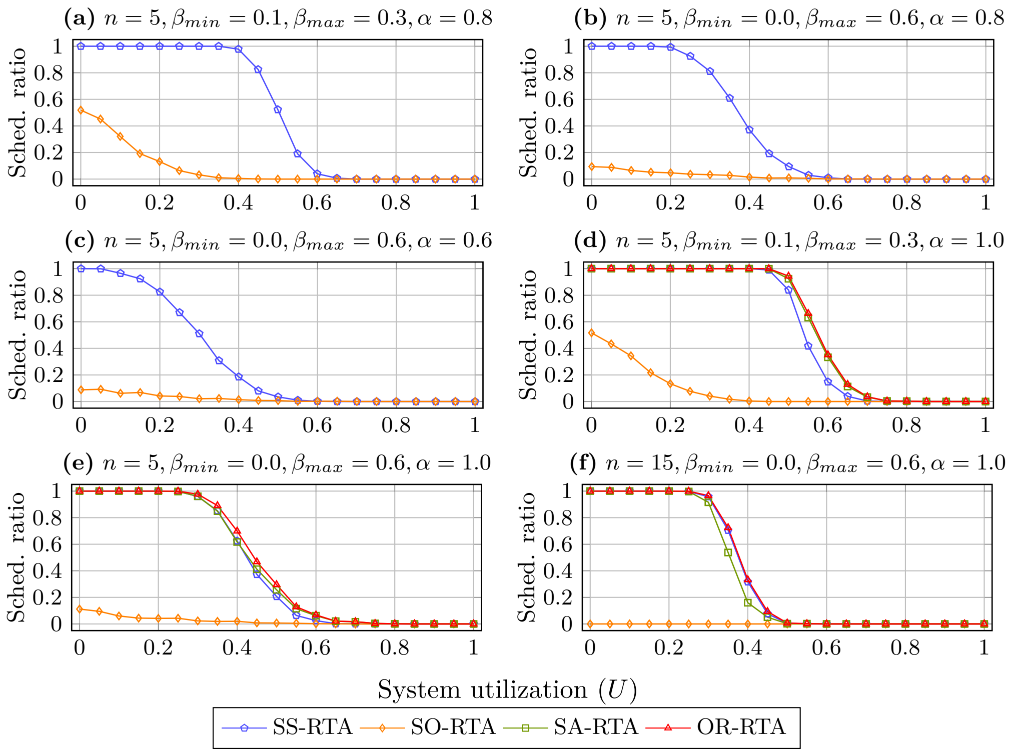


Figure 2 Comparison of the proposed RTA approach with state-of-the-art techniques in terms of the schedulability ratio obtained with different system configurations.

6 Conclusions and future work

This paper presented a response-time analysis for dynamic self-suspending tasks under preemptive EDF scheduling with constrained deadlines. The analysis is based on a model transformation to sporadic tasks with release jitter and on the subsequent application of a state-of-the-art analysis for the target task model. Experiments on randomly generated task sets compared the performance of the proposed approach in terms of schedulability ratio in the case of both implicit and constrained deadlines. The proposed approach significantly outperformed the baseline suspension-oblivious analysis in all the evaluated scenarios. Then, the approach was shown to provide comparable performance with the state-of-the-art response-time analysis for implicit deadlines by Günzel et al. [16]. Most importantly, the schedulability test which combines the two analyses was shown to outperform both techniques, meaning that the proposed approach does not dominate the response-time analysis by Günzel et al. [16] and vice-versa. Future work should consider leveraging the insights from both techniques to obtain a unifying analysis which can provide tighter WCRT upper bounds for the analysis of constrained-deadline self-suspending task systems under EDF. In addition, the proposed approach can be applied to the analysis of the EDD task model [1] towards the derivation of a response-time analysis for parallel tasks scheduled by the partitioned EDF algorithm and the analysis of hardware acceleration patterns under EDF scheduling.

References

1 Federico Aromolo, Alessandro Biondi, Geoffrey Nelissen, and Giorgio Buttazzo. Event-driven delay-induced tasks: Model, analysis, and applications. In *Proceedings of the 27th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2021)*, pages 53–65. IEEE, 2021.

2 Sanjoy K. Baruah, Louis E. Rosier, and Rodney R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2(4):301–324, 1990.

3 Enrico Bini and Giorgio C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

4 Alessandro Biondi, Alessio Balsini, Marco Pagani, Enrico Rossi, Mauro Marinoni, and Giorgio Buttazzo. A framework for supporting real-time applications on dynamic reconfigurable FPGAs. In *Proceedings of the 37th IEEE Real-Time Systems Symposium (RTSS 2016)*, pages 1–12. IEEE, 2016.

5 Björn B. Brandenburg. Multiprocessor real-time locking protocols. In Yu-Chu Tian and David Charles Levy, editors, *Handbook of Real-Time Computing*, pages 1–99. Springer, 2020.

6 Felipe Cerqueira, Geoffrey Nelissen, and Björn B. Brandenburg. On strong and weak sustainability, with an application to self-suspending real-time tasks. In *Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, pages 26:1–26:21, 2018.

7 Jian-Jia Chen, Geoffrey Nelissen, and Wen-Hung Huang. A unifying response time analysis framework for dynamic self-suspending tasks. In *Proceedings of the 28th Euromicro Conference on Real-Time Systems (ECRTS 2016)*, pages 61–71. IEEE, 2016.

8 Jian-Jia Chen, Geoffrey Nelissen, Wen-Hung Huang, Maolin Yang, Björn Brandenburg, Konstantinos Bletsas, Cong Liu, Pascal Richard, Frédéric Ridouard, Neil Audsley, Raj Rajkumar, and Georg von der Brüggen. Many suspensions, many problems: A review of self-suspending tasks in real-time systems. *Real-Time Systems*, 55(1):144–207, 2019.

9 UmaMaheswari C. Devi. An improved schedulability test for uniprocessor periodic task systems. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS 2003)*, pages 23–30. IEEE, 2003.

10 Zheng Dong and Cong Liu. Closing the loop for the selective conversion approach: A utilization-based test for hard real-time suspending task systems. In *Proceedings of the 37th IEEE Real-Time Systems Symposium (RTSS 2016)*, pages 339–350. IEEE, 2016.

11 Zheng Dong, Cong Liu, Soroush Bateni, Kuan-Hsun Chen, Jian-Jia Chen, Georg von der Brüggen, and Junjie Shi. Shared-resource-centric limited preemptive scheduling: A comprehensive study of suspension-based partitioning approaches. In *Proceedings of the 24th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2018)*, pages 164–176. IEEE, 2018.

12 Glenn A. Elliott, Bryan C. Ward, and James H. Anderson. GPUSync: A framework for real-time GPU management. In *Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS 2013)*, pages 33–44. IEEE, 2013.

13 Paul Emberson, Roger Stafford, and Robert I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *Proceedings of the 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*, pages 6–11, 2010.

14 José Fonseca, Geoffrey Nelissen, Vincent Nélis, and Luís Miguel Pinho. Response time analysis of sporadic DAG tasks under partitioned scheduling. In *Proceedings of the 11th IEEE Symposium on Industrial Embedded Systems (SIES 2016)*, pages 1–10. IEEE, 2016.

15 Laurent George, Nicolas Rivierre, and Marco Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. Research Report RR-2966, INRIA, France, 1996.

16 Mario Günzel, Niklas Ueter, and Jian-Jia Chen. Suspension-aware fixed-priority schedulability test with arbitrary deadlines and arrival curves. In *Proceedings of the 42nd IEEE Real-Time Systems Symposium (RTSS 2021)*, pages 418–430. IEEE, 2021.

17 Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen. Suspension-aware earliest-deadline-first scheduling analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):4205–4216, 2020.

18 Cong Liu and James H. Anderson. Suspension-aware analysis for hard real-time multiprocessor scheduling. In *Proceedings of the 25th Euromicro Conference on Real-Time Systems (ECRTS 2013)*, pages 271–281. IEEE, 2013.

19 Geoffrey Nelissen and Alessandro Biondi. The SRP resource sharing protocol for self-suspending tasks. In *Proceedings of the 39th IEEE Real-Time Systems Symposium (RTSS 2018)*, pages 361–372. IEEE, 2018.

20 Geoffrey Nelissen, José Fonseca, Gurulingesh Raravi, and Vincent Nélis. Timing analysis of fixed priority self-suspending sporadic tasks. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS 2015)*, pages 80–89. IEEE, 2015.

21 Marco Spuri. Analysis of deadline scheduled real-time systems. Research Report RR-2772, INRIA, France, 1996.

22 Georg von der Brüggen, Wen-Hung Huang, and Jian-Jia Chen. Hybrid self-suspension models in real-time embedded systems. In *Proceedings of the 23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2017)*, pages 1–9. IEEE, 2017.