Requirement-Based Analysis of Self-Suspending Tasks under EDF

Mario Günzel*, Federico Aromolo†, Alessandro Biondi†, Jian-Jia Chen*‡
**TU Dortmund University*, Dortmund, Germany, {mario.guenzel, jian-jia.chen}@tu-dortmund.de
‡*Lamarr Institute for Artificial Intelligence and Machine Learning*, Dortmund, Germany
†*Scuola Superiore Sant'Anna*, Pisa, Italy, {federico.aromolo, alessandro.biondi}@santannapisa.it

***Abstract*—While preemptive Earliest-Deadline-First (EDF) has been studied extensively in real-time systems, there are only few results when considering tasks with dynamic self-suspension behavior scheduled under EDF. Furthermore, all schedulability tests that have been developed in this context are based on analyzing specific intervals, hindering the performance of the analytical tightness of the result. In this work, we develop a schedulability test for EDF, built on a dynamic interval extension. That is, whenever the analysis cannot derive a decision to conclude the schedulability test, we iteratively extend the analysis interval to include additional carry-in jobs into the analysis. This is achieved by specifying execution-exceedance requirement for infeasibility of the system, i.e., by specifying how much workload must be accumulated within a certain time interval to achieve a deadline miss. Our approach outperforms all previous analyses and is the first to surpass the schedulability guarantees that can be provided for Deadline-Monotonic (DM) scheduling for dynamic self-suspending tasks, hence achieving a milestone in the analysis of EDF scheduling.**

***Index Terms*—real-time systems, schedulability analysis, EDF scheduling, self-suspending tasks, dynamic interval extension**

I. Introduction

Self-suspension refers to scenarios in which tasks yield their ready state despite being incomplete. Such behavior is found in many complex cyber-physical real-time systems, e.g., computation offloading to an external device, waiting on access rights for shared resources or data, and multiprocessor locking protocols [2], [7], [8]. Further applications can be found in the review paper by Chen et al. [10]. Since the debut made in 1988 [24], it is widely known that self-suspending tasks need a special treatment in the analysis. In the literature, schedulability tests for self-suspending tasks have been studied under different scheduling algorithms such as preemptive fixed-priority (FP) [3], [9], [11], [13], [15], [17]–[19], [22], Earliest-Deadline-First (EDF) [1], [12], [16], [21], and EDF-like (EL) [18], [26].

This paper considers the schedulability test of dynamic self-suspending sporadic real-time tasks under preemptive EDF on a single processor platform. Specifically, a dynamic self-suspending task may suspend arbitrarily often as long as the job's total suspension time does not exceed the specified maximum suspension time. There are four schedulability tests [1], [12], [16], [21] for EDF, in which Günzel et al. [16] and Aromolo et al. [1] focus on uniprocessor EDF and Dong and Liu [12] and Liu and Anderson [21] focus on global EDF in multiprocessor systems. For uniprocessor systems, Günzel et al. [16] show that the test by Dong and Liu [12] is the same as suspension-oblivious analysis (by considering suspension as computation) and the test by Liu and Anderson [21] is dominated by the test developed by Günzel et al. [16] in case all tasks self-suspend. All the known results of schedulability tests [1], [12], [16], [21] for dynamic self-suspending tasks under EDF are developed based on analyzing a specific analysis interval and none of them has been improved by considering more intervals in the analysis. This limitation to a specific analysis interval may hinder analytical performance.

To the best of our knowledge, this work is the first to achieve a *dynamic* interval extension procedure for self-suspending tasks under EDF. To achieve that, our approach formalizes *requirements for infeasibility*, specifying how much workload *must* be accumulated within a certain time interval for a deadline miss to happen. If a requirement for infeasibility within a time interval holds, it leads to system infeasibility; otherwise, if a requirement does not hold, it is not a valid requirement. Only in case it is not possible to validate whether a requirement for infeasibility holds or not due to the difficulty to quantify the number of carry-in jobs (i.e., those jobs that are released before but still execute during the analysis interval), we substitute the original requirement by a new set of requirements with expanded analysis intervals. During this interval expansion, we are able to determine the additional suspension that has to be considered for the analysis. Our approach continues testing the requirements and dynamically extending the analysis interval, until a schedulability decision is reached.

Contributions: In this work, we propose a requirement-based analysis approach for dynamic self-suspending tasks under preemptive EDF. Specifically, the contributions are as follows:

- The requirement-based analysis procedure is detailed and formalized in Section III. Since requirement-based analysis relies on a novel dynamic interval extension, we discuss its conceptual novelty in relation to known analysis approaches [1], [16] and the classical *demand bound function* approach [5] in Section IV.
- While the analysis is tunable with threshold values Θ, we propose a setting of the threshold in Section V.
- Our evaluation in Section VI shows that this requirement-

This paper has passed an Artifact Evaluation process. The artifact supporting this paper is available at https://doi.org/10.5281/zenodo.17203679 [14].

based approach outperforms previous analyses in schedulability ratio often by a large margin. Additionally, while prior suspension-aware analyses under preemptive EDF consistently perform worse than those under preemptive Deadline-Monotonic (DM) scheduling, our test is the first to provide schedulability guarantees for EDF that surpass those for DM found in the literature.

II. System Model

This work assumes a discrete time domain (i.e., using $\mathbb{Z}$), following the typical behavior of computing systems, which run with discrete time segments (i.e., system ticks).

Tasks and Jobs: We consider a task set $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ of n *sporadic* tasks with *dynamic self-suspending* behavior. Each task $\tau_i \in \mathbb{T}$ releases countably many jobs $\tau_i(j), j \in \mathbb{Z}_{\geq 1}$, and we specify the behavior of each task τ_i by three parameters

$$\tau_i \in \mathit{Spor}(T_i) \cap \mathit{WCET}(C_i) \cap \mathit{DynSus}(S_i). \tag{1}$$

That is, the minimum inter-arrival time $T_i \in \mathbb{Z}_{\geq 1}$ denotes that for each job $\tau_i(j), j \in \mathbb{Z}_{\geq 1}$ released at a time point $r_{i,j} \in \mathbb{Z}$ its subsequent job $\tau_i(j+1)$ is released no earlier than at $r_{i,j} + T_i$, the worst-case execution time (WCET) $C_i \in \mathbb{Z}_{\geq 1}$ is an upper bound on the amount of execution demand of each of its jobs, and the maximum suspension time $S_i \in \mathbb{Z}_{\geq 0}$ specifies that each job $\tau_i(j)$ of τ_i can suspend itself as long and as often as its maximum suspension time S_i is not exceeded. Additionally, each task $\tau_i \in \mathbb{T}$ has a relative deadline $D_i \in \mathbb{Z}_{\geq 1}$, which indicates that each job of τ_i must complete its execution within D_i upon its release. We consider *constrained-deadline* tasks in this work, i.e., $D_i \leq T_i$ for all $\tau_i \in \mathbb{T}$.

System Evolutions: Since the specification of tasks from Equation (1) only provides bounds for the execution, suspension, and release behavior, different execution-suspension patterns and different release patterns can be observed when running the system. The different system behaviors are formalized using the notion of *system evolutions*. A system evolution $\omega = (\rho^\omega, \chi^\omega)$ is composed of a release pattern ρ^ω and an execution-suspension pattern χ^ω. Formally, a release pattern ρ^ω is the collection the releases $r^\omega_{i,j}$ for each job $\tau_i(j)$, i.e., $\rho^\omega = (r^\omega_{i,j})_{\tau_i \in \mathbb{T}, j \in \mathbb{Z}_{\geq 0}}$, where inter-arrival time is respected, i.e., $r^\omega_{i,j+1} \geq r^\omega_{i,j} + T_i$ for all $\tau_i \in \mathbb{T}$ and $j \in \mathbb{Z}_{\geq 0}$. Furthermore, an execution-suspension pattern χ^ω specifies the sequence of execution and suspension segments $(c^\omega_{i,j,1}, s^\omega_{i,j,1}, c^\omega_{i,j,2}, \dots, s^\omega_{i,j,m^\omega_{i,j}-1}, c^\omega_{i,j,m^\omega_{i,j}}) \in \mathbb{Z}_{\geq 0}^{m^\omega_{i,j}}$ for each job $\tau_i(j)$. Formally, χ^ω is defined as $\chi^\omega = ((c^\omega_{i,j,1}, s^\omega_{i,j,1}, c^\omega_{i,j,2}, \dots, s^\omega_{i,j,m^\omega_{i,j}-1}, c^\omega_{i,j,m^\omega_{i,j}}))_{\tau_i \in \mathbb{T}, j \in \mathbb{Z}_{\geq 0}}$, with $\sum_{\xi=1}^{m^\omega_{i,j}} c^\omega_{i,j,\xi} \leq C_i$ and $\sum_{\xi=1}^{m^\omega_{i,j}-1} s^\omega_{i,j,\xi} \leq S_i$ for all $\tau_i \in \mathbb{T}$ and $j \in \mathbb{Z}_{\geq 0}$. We denote the set of all possible system evolutions ω of the task set $\mathbb{T}$ as Ω. Please note that although in a real system the observed system evolution might depend on the scheduling decisions made by the system, the system evolutions defined in this work do not need to capture these dependencies. Rather, for a safe analysis it is sufficient to aggregate all patterns Ω that can possibly occur.

Schedules: Dependent on the system evolution and algorithm for scheduling, different *schedules* can be observed. A schedule describes which task is executed at which time and can formally be defined by a function $\mathcal{S}\colon \mathbb{Z} \to \mathbb{T} \times \mathbb{Z}_{\geq 1} \cup \{\perp\}$ where $\mathcal{S}(t) = (\tau_i, j)$ denotes that the job $\tau_i(j)$ is executed during $[t, t+1)$, and $\mathcal{S}(t) = \perp$ denotes that no job is executed during $[t, t+1)$. A schedule fulfills the following conditions:

- No job executes before its release, i.e., for all $\tau_i \in \mathbb{T}$ and $j \in \mathbb{Z}_{\geq 0}$ we have $\mathcal{S}^{-1}(\tau_i, j) \cap (-\infty, r^\omega_{i,j}-1) = \emptyset$.
- No job executes more than specified by the system evolution ω, i.e., for all $\tau_i \in \mathbb{T}$ and $j \in \mathbb{Z}_{\geq 0}$ we have $|\mathcal{S}^{-1}(\tau_i, j)| \leq \sum_{\xi=1}^{m^\omega_{i,j}} c^\omega_{i,j,\xi}$.
- The schedule respects the suspension segments, i.e., between two execution segments $c^\omega_{i,j,\xi}$ and $c^\omega_{i,j,\xi+1}$, the job is not executed for $s^\omega_{i,j,\xi}$ time units.

We denote the set of all schedules $\mathcal{S}$ of a given system evolution $\omega \in \Omega$ as σ_ω. We say that a job *starts* its execution when it is executed for the first time, i.e., at $\min(\mathcal{S}^{-1}(\tau_i, j))$. Furthermore, we say that a job *finishes* when it reaches the demand specified in the system evolution ω, i.e., at the smallest $t \geq r^\omega_{i,j}$ such that $|\mathcal{S}^{-1}(\tau_i, j) \cap [r^\omega_{i,j}, t)| = \sum_{\xi=1}^{m^\omega_{i,j}} c^\omega_{i,j,\xi}$. If no such t exists, we say that the job starves, and for convenience say that the job finishes at ∞.

Scheduling Algorithms and Schedulability: This work tests the schedulability on one processor under *work-conserving preemptive EDF*, i.e., for schedules $\mathcal{S} \in \sigma_\omega$ where at each time the unfinished job with the earliest deadline (which is not currently suspending itself) is executed on the processor. Specifically, we say that $\mathbb{T}$ is schedulable under work-conserving preemptive EDF if, for all system evolutions $\omega \in \Omega$, all jobs meet their deadlines in the schedule $\mathcal{S} \in \sigma_\omega$ obtained by work-conserving preemptive EDF.

For the constructions to prove the correctness of the schedulability test presented in Section III, we impose more *relaxed* conditions (which are only adopted to ensure the correctness of our analysis). That is, we sometimes consider the more general *job-level fixed-piority (J-FP)* schedules where each job is assigned a priority level (which, in contrast to EDF, does not necessarily have to be derived from the absolute deadline) and at each time the job with the highest priority is executed. Furthermore, some jobs may be executed in a *non-work-conserving* manner, i.e., the job execution can be delayed even if the job does not suspend itself.

III. Requirement-Based Analysis

In this section, we present our proposed requirement-based schedulability analysis. Specifically, the approach formalizes requirements $(L, E) \in \mathbb{R}^2_{\geq 0}$, intuitively specifying that a certain amount of workload E *must* be accumulated inside an interval of length L to achieve a deadline miss (cf. Lemma 3). In multiple iterations, each requirement is tested (using Lemma 6) or extended into a larger set of requirements (using Lemma 9) until a decision is made. The full schedulability test is summarized in Algorithm 1.

Before we dive into the details of the schedulability analysis, we introduce the task set $\mathbb{T}' = \{\tau_1, \tau_2, \tau_3\}$ with parameters

Table I
PARAMETERS FOR THE RUNNING EXAMPLE $\mathbb{T}' = \{\tau_1, \tau_2, \tau_3\}$.

τ_i	T_i	C_i	S_i	D_i
τ_1	9	1	3	9
τ_2	15	3	8	15
τ_3	10	2	2	9

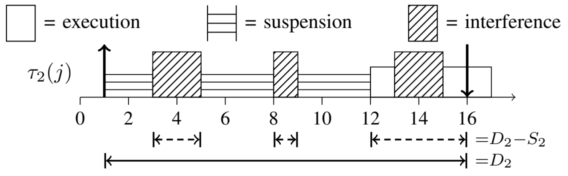


Figure 1. Deadline miss of a job $\tau_2(j)$ in the running example. Upward and downward facing arrows denote job release and deadline, respectively. During the interval of length D_2 more than $D_2 - S_2$ execution (indicated by dashed lines) has to be accumulated to achieve a deadline miss.

described in Table I to be used as a **running example** for this section. If this task set $\mathbb{T}'$ is unschedulable, then there exists a system evolution ω such that at least one of the jobs has a deadline miss in the schedule $\mathcal{S} \in \sigma_\omega$ obtained by preemptive EDF. In Figure 1, the deadline miss of a job $\tau_2(j)$ of task τ_2 is depicted. In the general case, to achieve a deadline miss for a job of τ_i, during an interval of length D_i more than $D_i - S_i$ execution has to be accumulated. For our running example, this results in three initial requirements: For $i = 1$: Accumulate more than $D_1 - S_1 = 9 - 3 = 6$ amount of execution inside an interval of length $D_1 = 9$. For $i = 2$: Accumulate more than $15 - 8 = 7$ amount of execution inside an interval of length 15. For $i = 3$: Accumulate more than $9 - 2 = 7$ amount of execution inside an interval of length 9. In the following we formalize such execution-exceedance requirements.

Definition 1 (Execution-Exceedance Requirements)**.** We define an *execution-exceedance requirement* by $(L, E) \in (\mathbb{R}_{\geq 0})^2$ with an interval length L and an exceedance value E. Intuitively, an execution-exceedance requirement asks whether within an interval of length L enough execution can be accrued to exceed the value E that is required to miss a deadline.

Formally, we use a *test function* $\mathcal{T}$ to test the requirement.

Definition 2 (Test for EDF)**.** We define a *test function for EDF* by $\mathcal{T} \colon (\mathbb{R}_{\geq 0})^2 \to \{\mathit{True}, \mathit{False}\}$ where $\mathcal{T}(L, E)$ is *True* if there exist a system evolution $\omega \in \Omega$, a schedule $\mathcal{S} \in \sigma_\omega$, and an interval $I = [a, b]$ of length $L = b - a$, such that the following two scheduling conditions **C1** and **C2** as well as the exceedance condition **EC** hold:

- **C1** $\mathcal{S}$ is obtained from work-conserving preemptive EDF.
- **C2** All jobs with deadline $< b$ have no deadline miss in $\mathcal{S}$.
- **EC** The amount of execution time after a by jobs with deadline $\leq b$ exceeds E.

Otherwise, the test returns $\mathcal{T}(L, E) = \mathit{False}$. We say that an execution-exceedance (L, E) requirement *holds* or is *fulfilled* under $\mathcal{T}$ if and only if $\mathcal{T}(L, E) = \mathit{True}$.

The idea of the requirement-based analysis is to specify a set of requirements $\mathcal{R}$ of which at least one must be satisfied to achieve a deadline miss. That is,

$$\mathbb{T} \text{ EDF unschedulable} \quad \Longrightarrow \quad \bigvee_{(L,E)\in\mathcal{R}} \mathcal{T}(L, E), \qquad (2)$$

where $\bigvee$ denotes the logical OR-operator. Hence, if all requirements $\mathcal{R}$ can be falsified, then the task set $\mathbb{T}$ must be schedulable under EDF. To determine and test the requirements $\mathcal{R}$, this section is organized as follows: In Section III-A, we determine an *initial* set of requirements $\mathcal{R}_0$. In Section III-B, the considered schedule is *relaxed* to enable more freedom for modifications of the schedule during the analysis and to integrate a tuning parameter Θ. Specifically, this section defines a test function $\mathcal{T}^*_\Theta$ for the relaxed schedule. In Section III-C, we detail an approach to *test* requirements under the relaxed test function $\mathcal{T}^*_\Theta$. In Section III-D, we consider requirements for which no decision with the test from Section III-C can be made. For those, we *substitute* the requirement by a new set of requirements with enlarged analysis intervals. In Section III-E, we control the size of the requirement set by identifying *dominated* requirements which can be safely removed. In Section III-F, the full schedulability test is concluded.

A. Initialization

We use the following initial set of execution-exceedance requirements $\mathcal{R}_0$.

Lemma 3 (Initiation)**.** *If $\mathbb{T}$ is unschedulable, then at least one of the execution-exceedance requirements $(D_i, D_i - S_i)$ with $\tau_i \in \mathbb{T}$ is fulfilled under the test $\mathcal{T}$ for EDF, i.e.,*

$$\mathbb{T} \textit{ EDF unschedulable} \quad \Longrightarrow \quad \bigvee_{(L,E)\in\mathcal{R}_0} \mathcal{T}(L, E) \qquad (3)$$

with $\mathcal{R}_0 := \{(D_i, D_i - S_i) \mid \tau_i \in \mathbb{T}\}$.

Proof: If $\mathbb{T}$ is unschedulable, then there exists a system evolution ω such that the schedule $\mathcal{S} \in \sigma_\omega$ obtained by preemptive EDF has at least one deadline miss. Let $\tau_i(j)$ be the job with the earliest deadline which has a deadline miss in $\mathcal{S}$, in which ties are broken arbitrarily. Then we consider the interval $I = [r^\omega_{i,j}, r^\omega_{i,j} + D_i]$. During this interval, the job $\tau_i(j)$ is suspended for up to S_i time units, and the remaining must be either execution from $\tau_i(j)$ or from jobs with higher priority, i.e., from jobs with deadline $\leq r^\omega_{i,j} + D_i$. The schedule $\mathcal{S}$ and interval I fulfill conditions **C1** and **C2** because of the construction of $\mathcal{S}$ and $\tau_i(j)$ being the earliest job with deadline miss. Furthermore, the amount of execution of jobs with deadline $\leq r^\omega_{i,j} + D_i$ exceeds $D_i - S_i$ because $\tau_i(j)$ experiences a deadline miss. Hence, **EC** is fulfilled with $E = D_i - S_i$. By Definition 2, $\mathcal{T}(D_i, D_i - S_i)$ is *True*, i.e., Equation (3) holds. ■

Based on Lemma 3, for our running example $\mathbb{T}'$,

$$\mathcal{R}_0 = \{(9{,}6), (15{,}7), (9{,}7)\} \text{ and} \qquad (4)$$

$$\mathbb{T}' \text{ EDF unschedulable } \Rightarrow \mathcal{T}(9, 6) \vee \mathcal{T}(15, 7) \vee \mathcal{T}(9, 7). \quad (5)$$

Now that we have requirements $\mathcal{R}_0$ for unschedulability of the task set, if we can show that all these requirements do *not* hold, then this means that the task set must be schedulable under EDF. Hence, Equation (3) serves as a schedulability test.

B. Relaxing the Schedule

To decide the schedulability of a task set in general, the requirement-based analysis uses a testing procedure (which determines whether a test returns *True* or *False*) in Section III-C, and a substitution procedure (which replaces the original requirement with a set of requirements with larger intervals in case no decision could be made with the test) in Section III-D. However, to control the length of the extended analysis interval, the substitution procedure relies on moving job releases to the latest possible time to produce a periodically released task.[1] When limiting our attention to EDF schedules only, a change of release times changes the absolute deadlines of jobs, which possibly results in a change of the priority order under EDF and unexpected changes in the schedule.

To mitigate this problem, we relax the conditions of the schedule used by the test function to consider arbitrary work-conserving job-level fixed-priority schedules.[2] Additionally, we also introduce a *relaxation threshold* Θ_i for each task $\tau_i \in \mathbb{T}$ which relaxes the work-conserving property for certain jobs. While the analysis is applicable with *any* choice of thresholds, they have an immediate impact on the number of task sets for which a decision can be made in Section III-C (specifically, Lemma 6) and hence on the number of interval extensions to be conducted in Section III-D. Therefore, the thresholds give us a lever to tune the schedulability test in Section V. The following formalizes the test function $\mathcal{T}^*_\Theta$ that is used to specify whether the requirement holds with the relaxed schedule and thresholds Θ.

Definition 4 (Test for Relaxed Schedule)**.** Consider thresholds $\Theta = (\Theta_1, \dots, \Theta_n)$ with $\Theta_i \in [0, D_i]$. We define a *test function for a relaxed schedule* by $\mathcal{T}^*_\Theta \colon (\mathbb{R}_{\geq 0})^2 \to \{\mathit{True}, \mathit{False}\}$ where $\mathcal{T}^*_\Theta(L, E)$ is *True* if there exists a system evolution $\omega \in \Omega$, a schedule $\mathcal{S} \in \sigma_\omega$, and an interval $I = [a, b]$ such that the following scheduling conditions **C1***–**C3*** and the exceedance condition **EC** from Definition 2 hold:

C1* $\mathcal{S}$ is a *job-level fixed-priority* schedule.

C2* In $\mathcal{S}$, all jobs $\tau_i(j)$ with release time $r^\omega_{i,j} < a - \Theta_i$ must be executed work-conservingly, and the jobs with release $r^\omega_{i,j} \geq a - \Theta_i$ can be executed *non-work-conservingly*.

C3* In $\mathcal{S}$, all jobs with deadline $\leq a$ have no deadline miss, and the jobs with deadline $> a$ are not constrained to meet their deadlines.

[1]This modification of release times is only an analytical procedure (namely, for the proof of Lemma 7) and does not have to be performed in the analyzed application. More specifically, the analyzed application does not need to allow the modification of release times to make our analysis applicable.

[2]Please note that this does not mean that the test is available to all work-conserving job-level fixed-priority scheduling algorithms. The reason is that important features of the EDF schedule are still preserved. For example, only jobs with deadline until the deadline of the job under analysis are considered for the accumulated workload.

We observe that conditions **C1***–**C3*** are less restrictive than conditions **C1**, **C2**. As such, $\mathcal{T}^*_\Theta$ can be used to replace $\mathcal{T}$, as formalized by the following lemma.

Lemma 5 (Relaxing the test function)**.** *Let* $\Theta \in \prod_{i=1}^n [0, D_i]$ *be task-specific thresholds for* $\mathbb{T}$*, then*

$$\bigvee_{(L,E)\in\mathcal{R}_0} \mathcal{T}(L, E) \quad \Rightarrow \quad \bigvee_{(L,E)\in\mathcal{R}_0} \mathcal{T}^*_\Theta(L, E) \tag{6}$$

with $\mathcal{R}_0$ *defined as in Lemma 3.*

Proof: Given that there exists $(L, E) \in \mathcal{R}_0$ such that $\mathcal{T}(L, E) = \mathit{True}$, there is a system evolution ω and a schedule $\mathcal{S} \in \sigma_\omega$ such that **C1**, **C2** and **EC** hold. Note that, when **C1** holds, also **C1***–**C3*** hold, given that (i) EDF is a JLFP scheduler (**C1***), (ii) **C2*** does not exclude work-conserving EDF scheduling after time $r^\omega_{i,j} \geq a - \Theta_i$, and (ii) **C3*** is unrelated to the scheduling algorithm. Furthermore, when **C2** holds, also **C1***–**C3*** hold, given that (i) **C1*** and **C2*** are unrelated to deadline misses, and (ii) **C3*** does not constrain deadline hits in $[a, b]$. Hence, $\mathcal{T}^*_\Theta(L, E) = \mathit{True}$ as well. ∎

For our running example, by using Lemma 5, the schedulability test from Equation (5) can be reformulated as:

$$\mathbb{T}' \text{ EDF unschedulable } \Rightarrow \mathcal{T}^*_\Theta(9,6) \vee \mathcal{T}^*_\Theta(15,7) \vee \mathcal{T}^*_\Theta(9,7) \tag{7}$$

For the remainder of Section III, we consider $\Theta = (0, 0, 0)$ for our running example $\mathbb{T}'$, meaning that carry-in jobs (i.e., those that are released before the analysis interval but still within it) are executed work-conservingly.

C. Test Requirements

In this section, we formulate sufficient conditions to determine whether a requirement $(L, E) \in \mathcal{R}$ is *True* or *False* under the test $\mathcal{T}^*_\Theta$. If the requirement is *False*, we can safely remove it from the set of requirements $\mathcal{R}$ to be considered. However, if the requirement is *True*, the task set is potentially unschedulable and hence a sufficient schedulability test based on it would stop with decision *Unknown*.

To test the requirement $(L, E) \in \mathcal{R}$, consider a task $\tau_i \in \mathbb{T}$ and an analysis interval $I = [a, b]$ with $b - a = L$. Considering only jobs with deadline $\leq b$, up to $\left\lfloor \frac{L+T_i-D_i}{T_i} \right\rfloor$ many jobs of τ_i have release times $\geq a$ and hence can execute after a. In addition to this number of jobs, since all jobs with deadline $\leq a$ have no deadline miss due to **C3***, at most one job arriving before a might be *carried in* and executed after a. That is, there might be a *carry-in job* for task τ_i when $(L + T_i - D_i) \bmod T_i > T_i - D_i$. We denote the indices of such tasks by

$$\mathbb{I}(L) := \{i \in \{1, \dots, n\} \mid (L + T_i - D_i) \bmod T_i > T_i - D_i\}\,. \tag{8}$$

Furthermore, given the thresholds $\Theta = (\Theta_1, \dots, \Theta_n)$, if we have $(L + T_i - D_i) \bmod T_i \geq T_i - \Theta_i$, then we know that the carry-in job can be executed *non-work-conservingly* in a relaxed schedule. Specifically, its execution can be pushed

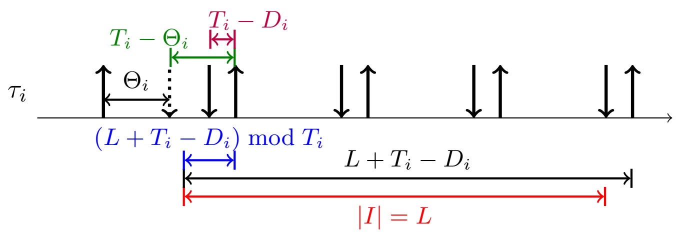


Figure 2. Illustration of the reasoning behind Equations (8) and (9). That is, there is possibly a carry-in job if $(L+T_i-D_i) \bmod T_i \geq T_i - D_i$, and that carry-in job can possibly be executed work-conservingly if $(L+T_i-D_i) \bmod T_i \geq T_i - \Theta_i$. For convenience of the presentation, only the deadline of the first and the latest job are depicted.

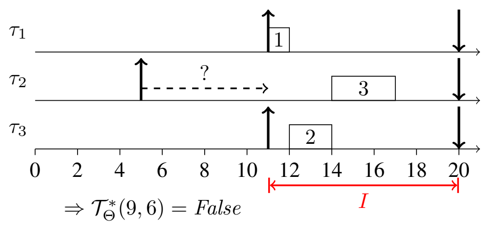


Figure 3. Testing requirement $(L, E) = (9, 6)$ in our running example $\mathbb{T}'$. The question mark indicates the possibility of τ_2's job being pushed depending on the schedule (i.e., interference and suspension). The sum of execution times inside the analysis interval I of length 9 *cannot exceed* 6, even with the potential carry-in job of τ_2. Therefore, $\mathcal{T}^*_\Theta(9, 6) = \mathit{False}$.

after a and we have to consider its full workload for **EC**. We define the indices of such tasks by $\mathbb{I}^*(L,\Theta) \subseteq \mathbb{I}(L)$ with

$$\mathbb{I}^*(L,\Theta) := \{i \in \mathbb{I}(L) \mid (L+T_i-D_i) \bmod T_i \geq (T_i-\Theta_i)\}\,. \tag{9}$$

Based on the previous observations, illustrated in Figure 2, we formalize our testing procedure using the following lemma. Please note that $\mathcal{T}^*_\Theta(L,E)$ might not be able to be determined by the lemma in case $\mathbb{I}(L) \neq \mathbb{I}^*(L,\Theta)$.

Lemma 6 (Testing Requirements). *Let (L, E) be an execution-exceedance requirement. If*

$$\sum_{i=1}^{n} \left\lfloor \frac{L+T_i-D_i}{T_i} \right\rfloor C_i + \sum_{i\in\mathbb{I}(L)} C_i \leq E, \tag{10}$$

then $\mathcal{T}^*_\Theta(L,E) = \mathit{False}$. Furthermore, given thresholds Θ, if

$$\sum_{i=1}^{n} \left\lfloor \frac{L+T_i-D_i}{T_i} \right\rfloor C_i + \sum_{i\in\mathbb{I}^*(L,\Theta)} C_i > E, \tag{11}$$

then $\mathcal{T}^*_\Theta(L,E) = \mathit{True}$.

Proof: We prove (10) by contradiction, assuming $\mathcal{T}^*_\Theta(L,E) = \mathit{True}$. Then, as this implies that **C3*** is satisfied, all jobs with deadline $\leq a$ cannot be executed after a, and the execution of jobs with deadline in $(a, b]$ has to be determined for **EC**. The number of jobs of τ_i with release $\geq a$ and deadline $\leq b$ is bounded by $k_i := \left\lfloor \frac{L+T_i-D_i}{T_i} \right\rfloor$, and since we consider constrained-deadline tasks, there can be at most one more job with deadline in $(a, b]$. The k_i-th latest job of τ_i with deadline $\leq b$ is released no later than $\gamma_i := b + T_i - D_i - k_i \cdot T_i = a + L + T_i - D_i - k_i \cdot T_i$ which is the same as $\gamma_i = a + (L + T_i - D_i) \bmod T_i$. Therefore, the $(k_i + 1)$-th latest job of τ_i with deadline $\leq b$ is released no later than $\gamma_i - T_i$ and has deadline no later than at $\gamma_i - T_i + D_i$. Checking whether $\gamma_i - T_i + D_i > a$ is equivalent to $(L + T_i - D_i) \bmod T_i > T_i - D_i$, i.e., $i \in \mathbb{I}(L)$. Hence, the workload from jobs with deadline in $(a, b]$ is bounded by $\sum_{i=1}^{n} k_i \cdot C_i + \sum_{i\in\mathbb{I}(L)} C_i$. Condition **EC** would then require this amount to exceed E, thereby reaching a contradiction.

For Equation (11), we show how to construct a relaxed schedule that guarantees conditions **C1***-**C3*** and **EC**. Consider an interval $I = [a, b]$ of length L and the JLFP schedule, thereby guaranteeing **C1***, where one job of each task τ_i is released at $b - D_i$ and sufficiently many preceding jobs are released as late as possible, i.e., respecting the minimum inter-arrival time T_i. In that case, $k_i = \left\lfloor \frac{L+T_i-D_i}{T_i} \right\rfloor$ is the number of jobs of task τ_i with deadline $\leq b$ and release $\geq a$. With a similar reasoning as above, the (k_i+1)-th latest job of τ_i, i.e., the latest job of τ_i released before a, is released at $\gamma_i - T_i$. This job is not constrained to meet its deadline if $\gamma_i - T_i > a - D_i$ by **C3***, which is equivalent to $i \in \mathbb{I}(L)$. The job can further be executed non-work-conservingly if $\gamma_i - T_i > a - \Theta_i$ by **C2***, which is satisfied if $i \in \mathbb{I}^*(L,\Theta)$. By pushing the execution of carry-in jobs within the interval I, we can ensure that each task τ_i with $i \in \mathbb{I}^*(L,\Theta)$ contributes an extra demand of C_i units that can be executed after a. In total, the amount of execution demanded within I is $\sum_{i=1}^{n} \left\lfloor \frac{L+T_i-D_i}{T_i} \right\rfloor C_i + \sum_{i\in\mathbb{I}^*(L,\Theta)} C_i$. If this value exceeds E, then **EC** holds, and we conclude $\mathcal{T}^*_\Theta(L,E) = \mathit{True}$. ■

Coming back to our running example, we have already formulated the initial requirements $\mathcal{R}_0 = \{(9,6), (15,7), (9,7)\}$ in Equation (4). We consider $(L, E) = (9, 6) \in \mathcal{R}_0$. Since Θ is set to $(0, 0, 0)$, following definitions of $\mathbb{I}$ and $\mathbb{I}^*$ yields $\mathbb{I}(9) = \{2\}$ and $\mathbb{I}^*(9,\Theta) = \emptyset$. Indeed, as depicted in Figure 3, for an analysis interval $I = [a, b]$, tasks τ_1 and τ_3 can each release $\left\lfloor \frac{9+T_1-D_1}{T_1} \right\rfloor = \left\lfloor \frac{9+T_3-D_3}{T_3} \right\rfloor = 1$ job with deadline $\leq b$ after a without an additional carry-in job, and for τ_2 we have no such job with release $\geq a$ since $\left\lfloor \frac{9+T_2-D_2}{T_2} \right\rfloor = 0$ but have one potential carry-in job. Calculating the left-hand side of Equation (10) with $L = 9$, we obtain $1{\cdot}1 + 0{\cdot}3 + 1{\cdot}2 + 3 = 6$, which relates to the total amount of execution after a of jobs with deadline $\leq b$ including potential carry-in jobs. Since this value does not exceed $E = 6$, the execution-exceedance requirement $(9, 6)$ does not hold, i.e., $\mathcal{T}^*_\Theta(9, 6) = \mathit{False}$. Therefore, we can remove this requirement from the schedulability test in Equation (7), leading to:

$$\mathbb{T}' \text{ EDF unschedulable } \Rightarrow \mathcal{T}^*_\Theta(15,7) \vee \mathcal{T}^*_\Theta(9,7) \tag{12}$$

Next, we continue by considering execution-exceedance requirement $(L, E) = (15, 7)$. As depicted in Figure 4, there are up to $\left\lfloor \frac{L+T_i-D_i}{T_i} \right\rfloor = 1$ jobs with release and deadline during I for each task $\tau_i \in \mathbb{T}'$. Only tasks τ_1 and τ_3 can have an additional carry-in job, i.e., $\mathbb{I}(15) = \{1, 3\}$. Calculating the left-hand side of Equation (10), we obtain

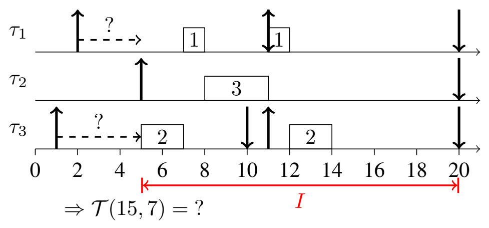


Figure 4. Testing requirement $(L, E) = (15, 7)$ in our running example $\mathbb{T}'$. With carry-in, the sum of execution time is 9. However, since the execution of carry-in jobs is unclear, only 6 time units execution can be guaranteed. Therefore, we cannot say whether $\mathcal{T}(15, 7)$ is *True* or *False*.

$(1 + 3 + 2) + (1 + 2) = 9$. Since $9 > E = 7$, we *cannot* use Equation (10) to derive that $\mathcal{T}^*_\Theta(L, E) = \mathit{False}$. Therefore, we consider Equation (11) next. Since $\Theta = (0, 0, 0)$, we obtain $\mathbb{I}^*(15, \Theta) = \emptyset$. Using that, we obtain $(1 + 3 + 2) + 0 = 6$ for the left-hand side of Equation (11), which is $\leq E = 7$. Hence, we *cannot* use Equation (11) neither to determine the outcome of $\mathcal{T}^*_\Theta(15, 7)$.

D. Substitution of Requirements

Motivated by the running example in Figure 4, we consider an execution-exceedance requirement $(L, E) \in \mathcal{R}$ for which the sufficient conditions from Lemma 6 do not suffice to make a decision, i.e., neither Equation (10) nor Equation (11) holds. For such requirements, we derive a set of new requirements that can be used to substitute (L, E). To derive new requirements, our strategy is to extend the analysis window to include additional carry-in jobs to refine our analysis. However, due to uncertainty in the sporadic release pattern and the execution-suspension pattern, many different schedules and release patterns have to be considered, making it non-trivial to find the right intervals for the extension. To that end, Lemma 7 limits the amount of possible schedules and determines a *critical task* $\tau_{i_{cr}}$ which is released periodically. This critical task serves as an anchor for the interval extension afterwards in Lemma 8.

Lemma 7 (Determine Critical task)**.** *Consider a task set* $\mathbb{T}$ *with task-specific thresholds* Θ *and an execution-exceedance requirement* (L, E) *such that* $\mathcal{T}^*_\Theta(L, E) = \mathit{True}$ *but Equation* (11) *does not hold. Then there exists a system evolution* ω*, a schedule* $\mathcal{S} \in \sigma_\omega$*, an analysis interval* $I = [a, b]$ *and a job* $\tau_{i_{cr}}(j_{cr})$ *with* $i_{cr} \in \mathbb{I}(L) \setminus \mathbb{I}^*(L, \Theta)$ *such that*

1) ***C1***–**C3*** and **EC** is fulfilled,*
2) $\tau_{i_{cr}}(j_{cr})$ *is still pending (i.e., not finished) at time* a *in* $\mathcal{S}$*, and*
3) $\tau_{i_{cr}}(j_{cr})$ *is released at* $r^\omega_{i_{cr},j_{cr}} = b - \left\lceil \frac{L+T_{i_{cr}}-D_{i_{cr}}}{T_{i_{cr}}} \right\rceil \cdot T_{i_{cr}} + T_{i_{cr}} - D_{i_{cr}}$.

We call such $\tau_{i_{cr}}$ *from this lemma a* critical task.

The proof of Lemma 7 is based on a construction (in the discrete time domain) of a concrete schedule which allows a critical task. To achieve that, job releases have to be actively postponed in the transformation. To avoid that the modification of job releases impacts the prioritization of jobs and hence the underlying schedule (as it would be the case with classical EDF), the relaxed scheduling conditions **C1***–**C3*** are substantial because they allow more freedom than EDF. Since the proof of Lemma 7 is rather technical, it is moved to Appendix A for readability.

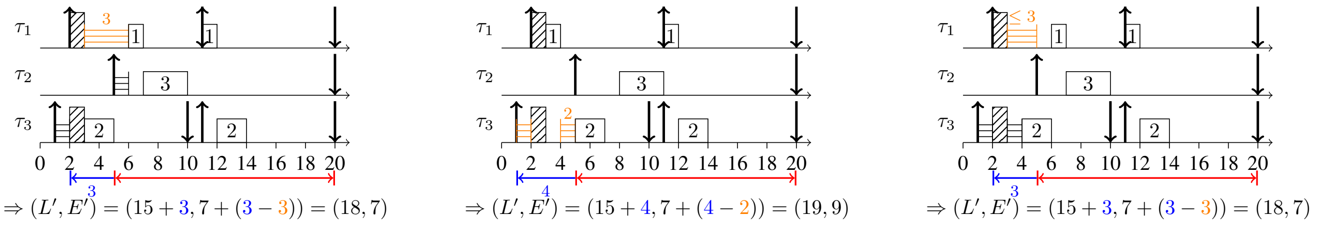
Using the critical task $\tau_{i_{cr}}$, we can extend the analysis interval. The rationale is that during the interval $[r^\omega_{i_{cr},j_{cr}}, a)$, whenever no job is executed, the carry-in job $\tau_{i_{cr}}(j_{cr})$ of the critical task $\tau_{i_{cr}}$ must suspend itself. Since $[r^\omega_{i_{cr},j_{cr}}, a)$ has a length of $\left\lceil \frac{L+T_{i_{cr}}-D_{i_{cr}}}{T_{i_{cr}}} \right\rceil \cdot T_{i_{cr}} - T_{i_{cr}} + D_{i_{cr}} - L$ and the carry-in job of the critical task has suspension of at most $S_{i_{cr}}$, the remaining $\max\left(\left\lceil \frac{L+T_{i_{cr}}-D_{i_{cr}}}{T_{i_{cr}}} \right\rceil \cdot T_{i_{cr}} - T_{i_{cr}} + D_{i_{cr}} - L - S_{i_{cr}}, 0\right)$ must be executed. Given that the execution-exceedance requirement (L, E) holds, it can be substituted by the execution-exceedance requirement $(\left\lceil \frac{L+T_{i_{cr}}-D_{i_{cr}}}{T_{i_{cr}}} \right\rceil \cdot T_{i_{cr}} - T_{i_{cr}} + D_{i_{cr}}, E + \max(\left\lceil \frac{L+T_{i_{cr}}-D_{i_{cr}}}{T_{i_{cr}}} \right\rceil \cdot T_{i_{cr}} - T_{i_{cr}} + D_{i_{cr}} - L - S_{i_{cr}}, 0))$. Formally, the execution-exceedance requirement can be substituted as follows.

Lemma 8 (Extension of Analysis Interval)**.** *Given that* $\mathcal{T}^*_\Theta(L, E) = \mathit{True}$*, but Equation* (11) *does not hold, we consider a critical task* $\tau_{i_{cr}}$ *from Lemma 7. Then,* $\mathcal{T}^*_\Theta(L'_{i_{cr}}, E'_{i_{cr}}) = $ *True holds with:*

$$\begin{aligned} L'_{i_{cr}} &= \left\lceil \frac{L + T_{i_{cr}} - D_{i_{cr}}}{T_{i_{cr}}} \right\rceil \cdot T_{i_{cr}} - T_{i_{cr}} + D_{i_{cr}} \\ E'_{i_{cr}} &= E + \max(L'_{i_{cr}} - L - S_{i_{cr}}, 0) \end{aligned} \tag{13}$$

Proof: We consider ω, $\mathcal{S}$, $I = [a, b]$, and $\tau_{i_{cr}}(j_{cr})$ provided by Lemma 7. Since $\tau_{i_{cr}}(j_{cr})$ is pending during $[r^\omega_{i_{cr},j_{cr}}, a)$ by 2), whenever the schedule idles in $[r^\omega_{i_{cr},j_{cr}}, a)$ the job $\tau_{i_{cr}}(j_{cr})$ must be suspended. This is possible for at most $\min(S_i, a - r^\omega_{i_{cr},j_{cr}})$ time units. Since further $a - r^\omega_{i_{cr},j_{cr}} = a - b + \left\lceil \frac{L+T_{i_{cr}}-D_{i_{cr}}}{T_{i_{cr}}} \right\rceil \cdot T_{i_{cr}} - T_{i_{cr}} + D_{i_{cr}} = L'_{i_{cr}} - L$ by 3), the total idle time in $[r^\omega_{i_{cr},j_{cr}}, a)$ is upper bounded by $\min(S_i, L'_{i_{cr}} - L)$. Since $[r^\omega_{i_{cr},j_{cr}}, a)$ has length $L'_{i_{cr}} - L$, the total execution time in $[r^\omega_{i_{cr},j_{cr}}, a)$ is lower bounded by $\max(L'_{i_{cr}} - L - S_{i_{cr}}, 0)$. By **EC**, the execution after a exceeds E, we conclude that the execution after $r^\omega_{i_{cr},j_{cr}}$ exceeds $E + \max(L'_{i_{cr}} - L - S_{i_{cr}}, 0) = E'_{i_{cr}}$. Since by increasing the analysis interval to $I = [r^\omega_{i_{cr},j_{cr}}, b]$ conditions **C1***–**C3*** are still satisfied, we conclude that ω and $\mathcal{S}$ fulfill all conditions **C1***–**C3*** and **EC** for the execution-exceedance requirement $(L'_{i_{cr}}, E'_{i_{cr}})$. Hence, $\mathcal{T}^*_\Theta(L'_{i_{cr}}, E'_{i_{cr}}) = \mathit{True}$. ∎

For our running example, in Section III-C, we already determined $(L, E) = (15, 7)$ to be an execution-exceedance requirement for which Lemma 6 is not sufficient to derive a decision. Different possible critical settings for (L, E) are illustrated in Figure 5, with a possible interference from earlier jobs during the interval $[2, 3]$. Specifically, in the all three critical settings, $I = [5, 20]$ is marked red, and the additional part for the interval extension is marked blue. Figure 5a shows a critical setting where only τ_1 has a carry-in job (that is still pending at the beginning of I). Hence, $i_{cr} = 1$, and the new requirement (L', E') can be computed using Lemma 8 as $(L', E') = (18, 7)$. Figure 5b shows a critical setting



τ_1
τ_2
τ_3
0 2 4 6 8 10 12 14 16 18 20
$\Rightarrow (L', E') = (15+3, 7+(3-3)) = (18,7)$
$\Rightarrow (L', E') = (15+4, 7+(4-2)) = (19,9)$
$\Rightarrow (L', E') = (15+3, 7+(3-3)) = (18,7)$

(a) Carry-in: τ_1 only. (b) Carry-in: τ_3 only. (c) Carry-in: Both τ_1 and τ_3.

Figure 5. Expansion procedure for different carry-in scenarios for the running example $\mathbb{T}'$ with execution-exceedance requirement $(L, E) = (15, 7)$. Due to the expansion procedure, the requirement $(15, 7)$ is replaced by $(18, 7)$ and $(19, 9)$.

where only τ_3 has a carry-in job. Here, $i_{cr} = 3$ and the interval extension yields $(L', E') = (19, 9)$. In Figure 5c, both tasks τ_1 and τ_3 have a carry-in job. Here, the task with the lowest-priority carry-in job is the critical task, i.e., $i_{cr} = 1$, which results in the new requirement $(L', E') = (18, 7)$. We see from this example that different tasks τ_i with index $i \in \mathbb{I}(L) \setminus \mathbb{I}^*(L, \Theta)$ can potentially become critical tasks. Hence, we need to investigate all of them. This leads to the following lemma for substitution.

Lemma 9 (Substitution). *Let (L, E) be an execution-exceedance requirement such that Equation* (11) *does not hold. Then one of the requirements $(L'_{i_{cr}}, E'_{i_{cr}})$ defined by Equation* (13) *with $i_{cr} \in \mathbb{I}(L) \setminus \mathbb{I}^*(L, \Theta)$ must be satisfied if $\mathcal{T}^*_\Theta(L, E)$ is True. Formally, we have:*

$$\mathcal{T}^*_\Theta(L, E) \Rightarrow \bigvee_{i_{cr} \in \mathbb{I}(L) \setminus \mathbb{I}^*(L,\Theta)} \mathcal{T}^*_\Theta(L'_{i_{cr}}, E'_{i_{cr}}) \tag{14}$$

Proof: If $\mathcal{T}^*_\Theta(L, E) = \mathit{False}$, then the implication of Equation (14) is trivially correct. Otherwise, if $\mathcal{T}^*_\Theta(L, E) = \mathit{True}$, then there exists a critical task $\tau_{i_{cr}}$ with $i_{cr} \in \mathbb{I}(L) \setminus \mathbb{I}^*(L, \Theta)$ by Lemma 7 and further $\mathcal{T}^*_\Theta(L'_{i_{cr}}, E'_{i_{cr}}) = \mathit{True}$ by Lemma 8. We conclude that Equation (14) is a correct implication in both cases. ■

Using this substitution for our running example we achieve $\mathcal{T}^*_\Theta(15, 7) \Rightarrow \mathcal{T}^*_\Theta(18, 7) \vee \mathcal{T}^*_\Theta(19, 9)$. Using this substitution for the set of requirements $\mathcal{R} = \{(15,7), (9,7)\}$, we obtain $\mathcal{R} = \{(18,7), (19,9), (9,7)\}$. This changes the schedulability test from Equation (12) to:

$$\mathbb{T}' \text{ EDF unschedulable } \Rightarrow \mathcal{T}^*_\Theta(18, 7) \vee \mathcal{T}^*_\Theta(19, 9) \vee \mathcal{T}^*_\Theta(9, 7) \tag{15}$$

After substitution, our approach continues testing and substituting the requirements in $\mathcal{R}$ until a schedulability decision can be reached or until the number of iterations exceeds a predefined bound.

E. Reducing the Space of Requirements

To avoid that $\mathcal{R}$ (by applying the substitution mechanism in Lemma 9) grows too large, we identify *dominated* requirements. Specifically, a requirement $(L_1, E_1) \in \mathcal{R}$ is dominated by another requirement $(L_2, E_2) \in \mathcal{R}$ if (L_1, E_1) can only be satisfied if (L_2, E_2) is satisfied. In that case, the requirement (L_1, E_1) can safely be removed from $\mathcal{R}$ without any impact on the analysis.

One such example of dominated requirements can be found in the set of requirements $\mathcal{R} = \{(18,7), (19,9), (9,7)\}$ of our running example. That is, if $(L_1, E_1) = (9, 7)$ is satisfied, i.e., for an analysis interval $I = [b-9, b]$ of length 9 more than 7 time units of execution can be accumulated, then these 7 time units can also be accumulated for an analysis interval $I = [b-18, b]$ of length 18, i.e., $(L_2, E_2) = (18, 7)$ is satisfied. Hence, $(9, 7)$ is dominated by $(18, 7)$ and can safely be removed from $\mathcal{R}$. The following lemma provides a general rule to identify dominated execution-exceedance requirements.

Lemma 10 (Dominance of Requirements)**.** *Given two execution-exceedance requirements (L_1, E_1) and (L_2, E_2), if $E_2 \leq E_1$ and $L_2 \geq L_1$, then $\mathcal{T}^*_\Theta(L_1, E_1) \Rightarrow \mathcal{T}^*_\Theta(L_2, E_2)$. We say that (L_1, E_1) is* dominated *by (L_2, E_2).*

Proof: If $\mathcal{T}^*_\Theta(L_1, E_1) = \mathit{True}$, then by Definition 4 there exist ω, $\mathcal{S}$ and $I = [a, b]$ with length $b-a = L_1$ such that **C1***–**C3*** and **EC** hold. In particular, the accumulated execution after a exceeds E_1, jobs of τ_i released before $a - \Theta_i$ are executed work-conservingly, and jobs with deadline $\leq a$ have no deadline miss. Since $b - L_2 \leq a$, this also means that the accumulated execution after $b - L_2$ exceeds E_1 (which is $\geq E_2$), jobs of τ_i released before $b - L_2 - \Theta_i$ are executed work-conservingly, and jobs with deadline $\leq b - L_2$ have no deadline miss. Hence, for the same ω and $\mathcal{S}$ but with analysis interval $I = [b - L_2, b]$, the execution-exceedance requirement (L_2, E_2) is satisfied for test $\mathcal{T}^*_\Theta$, i.e., $\mathcal{T}^*_\Theta(L_2, E_2) = \mathit{True}$ ■

For the two requirements $(L_1, E_1) = (9, 7)$ and $(L_2, E_2) = (18, 7)$ from $\mathcal{R}$, indeed Lemma 10 determines that $(9, 7)$ is dominated by $(18, 7)$. Hence, by utilizing the dominance, we can remove $(9, 7)$ from $\mathcal{R}$. Hence, the schedulability test is:

$$\mathbb{T}' \text{ EDF unschedulable } \Rightarrow \mathcal{T}^*_\Theta(18, 7) \vee \mathcal{T}^*_\Theta(19, 9) \tag{16}$$

Our approach continues testing and substituting execution-exceedance requirements until a decision is made. The dominance of requirements is tested each time new requirements are added in the substitution process.

F. Concluding the Schedulability Test

In the previous subsections, we have derived an initialization of the set of requirements $\mathcal{R} = \mathcal{R}_0$ in Lemma 3, a test of requirements in Lemma 6, a substitution of requirements in Lemma 9, and a dominance of requirements in Lemma 10. Using these results, the full procedure of our schedulability test is summarized in Algorithm 1. The schedulability test is sufficient and therefore returns either *Feasible* or *Unknown*. Besides the task set $\mathbb{T}$, a set of task-specific thresholds $\Theta \in \prod_{i=1}^{n}[0, D_i]$, and a maximal number of iterations $M \in \mathbb{Z}_{\geq 0} \cup \{\infty\}$ is specified. While the analysis is correct for all values of Θ and M, heuristics to choose Θ are discussed in Section V, and $M = \infty$ yields the best result if runtime of the analysis is not a concern. By collecting the lemmas from the previous sections, we can prove the correctness of Algorithm 1 as a sufficient schedulability test.

Algorithm 1 Requirement-Based Schedulability Test

```
Input: - $\mathbb{T}$ Task set
       - $\Theta$ Thresholds                 ▷ Use heuristics from Section V
       - $M$ Maximal number of iterations
Output: Feasible or Unknown
 1: Initialize $\mathcal{R} := \mathcal{R}_0$                  ▷ Lemma 3
 2: while $\mathcal{R}$ is not empty do
 3:     if Number of iterations exceeds $M$ then
 4:         return Unknown
 5:     Select and remove $(L, E)$ with smallest $L$ from $\mathcal{R}$
 6:     if Equation (10) holds for $(L, E)$ then
 7:         continue                               ▷ $(L, E)$ is False
 8:     if Equation (11) holds for $(L, E)$ then
 9:         return Unknown                ▷ One requirement is True
10:     for each $i_{cr} \in \mathbb{I}(L) \setminus \mathbb{I}^*(L, \Theta)$ do
11:         Add $(L'_{i_{cr}}, E'_{i_{cr}})$ from Equation (13) to $\mathcal{R}$
12:     Remove dominated requirements in $\mathcal{R}$ by Lemma 10
13: return Feasible                       ▷ All requirements are False
```

Theorem 11 (Correctness of Test)**.** *The schedulability test for preemptive EDF presented in Algorithm 1 is correct.*

Proof: It is to be shown that at the beginning and at the end of each iteration of the while loop

$$\mathbb{T} \text{ EDF unschedulable} \quad \Longrightarrow \quad \bigvee_{(L,E)\in\mathcal{R}} \mathcal{T}^*_\Theta(L, E) \tag{17}$$

holds. For the initialization in line 1, this is correct due to a combination of Lemma 3 and Lemma 5. Hence, when starting the while loop in line 2, Equation (17) holds. During one iteration, if Equation (10) holds in line 6, then $\mathcal{T}^*_\Theta(L, E) = \mathit{False}$, as shown by Lemma 6, and (L, E) can safely be removed while preserving the correctness of Equation (17). Furthermore, if Equation (11) holds in line 8, then $\mathcal{T}^*_\Theta(L, E) = \mathit{True}$ by Lemma 6, and our test terminates with decision *Unknown*. Otherwise, we replace (L, E) using the substitution in line 10, and remove dominated requirements in line 12, which both preserve Equation (17) by Lemmas 9 and 10, respectively. This proves the correctness of Equation (17) after each iteration. The algorithm returns *Feasible* if $\mathcal{R} = \emptyset$, which is only possible if all requirements are shown to be unsatisfied under $\mathcal{T}^*_\Theta$, i.e., if the right-hand side of Equation (17) is *False*. ■

Time Complexity: In the following we discuss the worst-case time complexity of the proposed Algorithm 1 distinguishing the case $M = \infty$ and $M < \infty$. **Case** $M = \infty$**:** If not controlled by the maximum number of iterations M, the analysis could potentially continue refining indefinitely, and the worst-case time complexity is unbounded. However, the algorithm can only refine indefinitely if line 10 in Algorithm 1 can be reached infinitely many times. This is a corner case in which for every requirement set $\mathcal{R}$, there is at least one execution exceedance requirement, for which Lemma 6 is unable to make a decision (i.e., both Equations (10) and (11) do not hold). Please note that in the evaluation in Section VI, we run Algorithm 1 with $M = \infty$ and never reach a case where the algorithm does not terminate naturally. **Case** $M < \infty$**:** If M is set to a finite number by the designer, then Algorithm 1 runs at most M iterations. In each iteration, checking Equations (10) and (11) in lines 6 and 8, respectively, of Algorithm 1 has a complexity of $\mathcal{O}(n)$ (where n is the number of tasks). Furthermore, one interval is removed from $\mathcal{R}$, and up to n new execution-exceedance requirements can be generated in line 10 of Algorithm 1 and added into $\mathcal{R}$. Checking for dominated requirements in line 12 of Algorithm 1 can be done by checking whether the newly added n intervals are dominated by the $|\mathcal{R}|$ intervals of $\mathcal{R}$ (and vice versa), which has a time complexity of $\mathcal{O}(|\mathcal{R}|n)$. Since after M iterations there are not more than $M \cdot n$ elements in $\mathcal{R}$, the worst-case time complexity for line 12 of Algorithm 1 is $\mathcal{O}(M \cdot n^2)$. Hence, the total worst-case time complexity with given M and n is $\mathcal{O}(M^2 \cdot n^2)$. We note that this is only a naive upper bound and that it could potentially be tightened by using for example more dedicated bounds on $|\mathcal{R}|$ or more efficient implementation to check for dominated requirements in line 12 of Algorithm 1.

IV. Dynamic Interval Extension in EDF Analysis

We discuss the conceptual novelty of our requirement-based analysis in comparison to the state-of-the-art approaches for uniprocessor preemptive EDF.[3]

a) Analysis by Günzel et al. [16]: The work by Günzel et al. provides two methods. The utilization-based approach identifies redundant self-suspension to improve the classical suspension-oblivious test. Here, the analysis is statically chosen to be the suspension-aware busy-interval, and no dynamic interval extension is used at all. The response time analysis examines jobs $\tau_i(j)$ of task τ_i by determining the interference of higher priority jobs on the interval $[r^\omega_{i,j}, r^\omega_{i,j}+D_i)$. Although it relies on an iterative approach to update the determined bounds on the worst-case response time, there is no interval extension involved to make the analysis tighter.

b) Analysis by Aromolo et al. [1]: The approach by Aromolo et al. translates the self-suspending task set into a task set with jitter and then applies the results by Spuri [25]. The analysis by Spuri relies on the observation that the worst-case response time is found in a busy period with a specific release pattern. Hence, the analysis interval is *fixed* to the busy period and all jobs inside this busy period have to be checked. This is structurally very different to the dynamic interval extension developed in this work, where additional jobs are

[3]The test by Dong and Liu [12] and the test by Liu and Anderson [21] are not discussed because they are dominated when considering *uniprocessor* EDF with dynamic self-suspending tasks [16].

only included in the analysis interval if no schedulability decision can be derived for the current analysis interval.

c) Classical Demand-Based Analysis [5]: For ordinary sporadic real-time tasks under preemptive EDF, the schedulability can be tested using the *demand bound function* (dbf) proposed by Baruah et al. [5]. The proof of the dbf-based schedulability test can be sketched as follows: Suppose that a job of task τ_k is the first job which misses its deadline under preemptive EDF at time d_k. It can be shown that the EDF schedule is busy and executes jobs whose absolute deadlines are $\leq d_k$ from a certain time t_0 to d_k. Furthermore, it can be proven that only jobs arriving at or after time t_0 are executed in this interval of time. Therefore, the deadline miss of the job implies that the demand of the jobs whose arrival time is at or after t_0 and absolute deadline $\leq d_k$ is strictly more than $d_k - t_0$. By contrapositive, *for all possible intervals from the release time to the absolute deadline of* any *two jobs*, it has to be tested whether the demand (i.e., the accumulated execution time of the jobs that arrive and are due in this interval) exceeds the interval length. Informally speaking, the unschedulability of EDF implies the *existence of a time interval* in which the required demand is strictly more than the interval length. The contrapositive nature mathematically results in a test of all relevant interval lengths.[4] To the best of our knowledge, for dynamic self-suspending sporadic real-time tasks, there is no schedulability test for preemptive EDF using similar constructs like demand bound functions. One potential reason is the difficulty to clearly define t_0 and the demand that must be safely included when considering self-suspending tasks.

Our requirement-based analysis follows a completely different concept. We analyze the requirement that further includes carry-in jobs to make the system infeasible in an interval and *further intentionally expand the interval if necessary* due to the included carry-in jobs to validate the assertion of infeasibility. In other words, our requirement-based analysis establishes a new analysis paradigm, which *actively* expands its intervals of interest due to the updated requirements for infeasibility.

V. Tuning of the Threshold Parameter

While any choice of thresholds $\Theta = (\Theta_1, \dots, \Theta_n) \in \prod_{i=1}^{n}[0, D_i]$ is safe to be used for the schedulability test presented in Section III, the choice leaves us with a tuning parameter for the analysis. In this section, we discuss different options and heuristics to choose Θ.

Obvious choices for Θ are to choose the minimal values $\Theta = \Theta^{min}$ or the maximal values $\Theta = \Theta^{max}$, with:

$$\Theta^{min} := (0, \dots, 0) \qquad \Theta^{max} := (D_1, \dots, D_n) \tag{18}$$

While $\Theta = \Theta^{min}$ means that no additional carry-in jobs have to be automatically considered for the testing procedure in Lemma 6, i.e., the set $\mathbb{I}^*(L, \Theta)$ is empty in Equation (11), this does also mean that for each possible carry-in job an interval extension has to be considered in the substitution process in Lemma 9 (i.e., the set $\mathbb{I}(L) \setminus \mathbb{I}^*(L, \Theta)$ is the same as $\mathbb{I}(L)$). On the other hand, for $\Theta = \Theta^{max}$ we have $\mathbb{I}^*(L, \Theta) = \mathbb{I}(L)$, meaning that we always assume every possible carry-in job executed to its WCET and never perform interval expansion.

To come up with a reasonable choice of Θ, we have to assess whether the pessimism from substitution outweighs the pessimism of considering the carry-in job being executed with its worst-case execution time. Our strategy for the heuristic is to avoid substitution when such a scenario of full carry-in will be considered. For a task τ_i and analysis interval $I = [a, b]$ this is the case if the suspension plus the interference on the suspension of the potential carry-in job of τ_i pushes the start of the carry-in job beyond time a. We know by definition that the suspension of τ_i is bounded by S_i. Furthermore, the impact of interference on the suspension S_i can be roughly estimated by enlarging the suspension by the factor $1/(1-(U_\mathbb{T}-U_i))$, where $U_i = C_i/T_i$ is the utilization of τ_i and $U_\mathbb{T}$ is the utilization of the task set $\mathbb{T}$, i.e., $U_\mathbb{T} = \sum_{\tau_i \in \mathbb{T}} U_i \leq 1$. Hence, the carry-in job can start after a if it is released no earlier than at $a - S_i/(1-(U_\mathbb{T}-U_i))$. This results in a suspension-aware heuristic $\Theta^{sus} := (\Theta_1^{sus}, \dots, \Theta_n^{sus})$ with:

$$\Theta_i^{sus} := \min\left(D_i,\, S_i/(1 - (U_\mathbb{T} - U_i))\right) \tag{19}$$

We note that although $S_i/(1 - (U_\mathbb{T} - U_i))$ is only a rough estimation of the suspension and interference of task τ_i, the analysis remains valid for any choice of thresholds Θ. Hence, this estimation is sufficient to derive a safe test, and more precise estimation might yield even better results.

Another aspect to consider is that jobs with very low execution time are very cheap to include as carry-in and hence we would like to favor such behavior. One way to achieve this is to use a factor $(1 + (1 - \frac{C_i}{\max_{\tau_j \in \mathbb{T}} C_j})^n)$ when calculating the threshold Θ_i. Intuitively, the closer the WCET of task τ_i is to the maximal WCET, the more we rely on suspension-aware heuristics, and if the WCET of τ_i is very small (i.e., close to zero) we double the threshold. The parameter n accounts for the fact that the more tasks are in the set, the more likely is it that the maximal and minimal WCET deviate a lot, which without regulation would negatively affect tasks with medium WCET. The modification of Θ^{sus} results in the suspension-and-execution-aware heuristic $\Theta^{sus,exec} := (\Theta_1^{sus,exec}, \dots, \Theta_n^{sus,exec})$ with:

$$\Theta_i^{sus,exec} := \min\left(D_i,\, \frac{S_i}{1 - (U_\mathbb{T} - U_i)} \cdot (1 + (1 - \frac{C_i}{\max_{\tau_j \in \mathbb{T}} C_j})^n)\right) \tag{20}$$

VI. Evaluation

A. Experimental Setup

In the experiments, the evaluated analysis methods are applied to randomly generated task sets, under various configurations of the task set generator. The implementation of the experiments is provided as an artifact [14].

The number of tasks n generated for each task set and the utilization U of each generated task set are varied within the

[4]There is also an extension of this general concept to multiprocessor [4]. While their work considers multiple extension points, no dynamic interval extension is conducted. Similar concepts for fixed-priority scheduling have been studied as well [20].

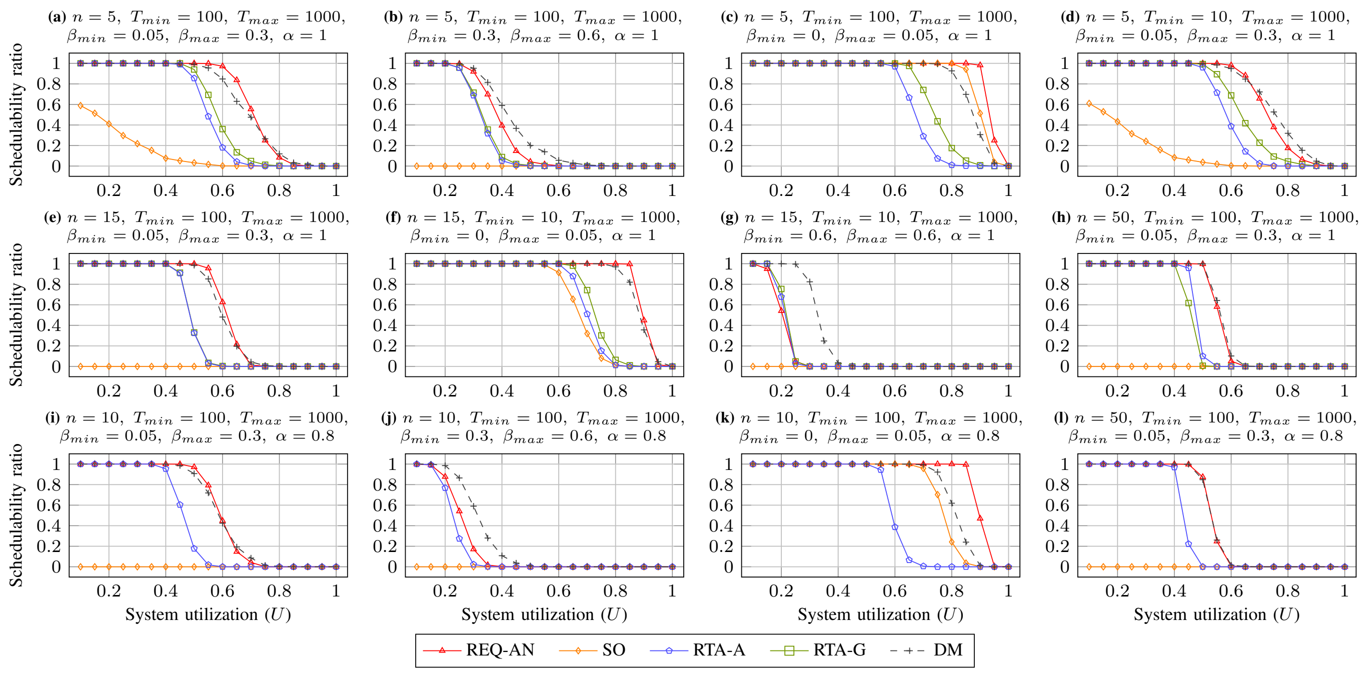


Figure 6. Schedulability ratios obtained with different configuration parameters of the task set generation procedure.

experiments. In the generation of a task set $\mathbb{T}$, the UUniFast algorithm [6] was applied to generate the utilization U_i for each task in $\mathbb{T}$, such that $U = \sum_{\tau_i \in \mathbb{T}} U_i$. The minimum inter-arrival time T_i of each task τ_i in $\mathbb{T}$ 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 values of T_i. 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 $\beta_{min} \leq \beta_{max} \in [0, 1]$. 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. Note that $\alpha \in [0, 1]$ produces constrained deadlines ($D_i \leq T_i$), whereas $\alpha = 1$ generates implicit deadlines ($D_i = T_i$). The experiments are performed by varying the system utilization U from 0.1 to 1 in increments of 0.05, and generating 1000 task sets for each value of U. The analysis approaches were then applied to each of the generated task sets in order to assess the schedulability ratio achieved by each method with respect to a specific system utilization U and a given generation configuration, that is, the ratio of task sets deemed schedulable by a specific analysis over the number of task sets generated for that system utilization point U under that configuration. The following analysis strategies have been assessed:

- **REQ-AN**: Our proposed requirement-based analysis for EDF scheduling, using the schedulability test in Algorithm 1 with $M = \infty$. We tested the setting of Θ by adopting Equations (18), (19), and (20). We only report the results based on Equation (20), as it provided the best performance in our evaluation.
- **SO**: Suspension-oblivious analysis for EDF, where suspensions are regarded as additional computation time (i.e., setting the WCETs of each task τ_i to $C_i + S_i$ and suspension to 0). For implicit deadlines, it is checked whether the total (inflated) utilization is less than or equal to 1 [23]. For constrained deadlines, the response-time analysis for sporadic tasks under EDF by Spuri [25] is applied to the task set with inflated WCETs.
- **RTA-G**: Suspension-aware response-time analysis for EDF scheduling by Günzel et al. [16] (see Section IV), compatible with task sets with implicit deadlines only.
- **RTA-A**: Suspension-aware response-time analysis for EDF scheduling by Aromolo et al. [1] (see Section IV).
- **DM**: Suspension-aware response-time analysis for Deadline Monotonic (DM) scheduling, which assigns higher fixed priorities to tasks with smaller relative deadlines. We consider a combination of the unifying analysis by Chen et al. [9] (using the three main heuristics of the unifying analysis framework, namely jitter-based analysis, blocking-based analysis, and the linearized unifying analysis [9]) and the improved jitter-based analysis by Günzel et al. [17]. A task set was deemed schedulable if at least one of these analysis classified it as such.

Note that **DM** considers fixed-priority scheduling rather than EDF scheduling. This approach is included in the experimental comparison to better highlight the performance gain attained by the requirement-based analysis, which is shown to be the first EDF analysis for self-suspending tasks that can consistently reach or even surpass the schedulability performance of the existing fixed-priority analyses for self-suspending tasks.

B. Evaluation Results

Implicit Deadlines: Experimental results for the case of implicit deadlines ($\alpha = 1$) are provided in Figures 6(a)-(h).

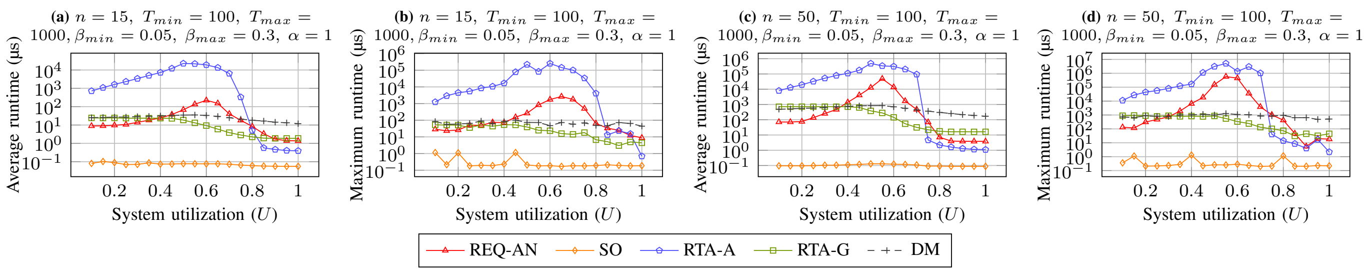


Figure 7. Analysis runtimes obtained with different configuration parameters of the task set generation procedure.

The configuration parameters are reported above each plot. Figure 6(a) reports the results obtained with a limited number of tasks ($n = 5$) and moderate suspension lengths ($\beta_{min} = 0.05,\ \beta_{max} = 0.3$), considering periods in the range $[T_{min} = 100,\ T_{max} = 1000]$. As expected, the results show decreasing schedulability performance with higher values of the system utilization for all the evaluated techniques. With this configuration, the proposed approach (**REQ-AN**) outperforms the existing suspension-oblivious (**SO**) and suspension-aware (**RTA-A**, **RTA-G**) analysis techniques for EDF scheduling by a large margin, achieving a schedulability performance level comparable to that of the fixed-priority analysis (**DM**). When longer suspensions are considered (Figure 6(b)), **REQ-AN** still outperforms the existing techniques for EDF, **RTA-A** and **RTA-G**, albeit by a smaller margin, and performs marginally worse than **DM**. With short suspensions (Figure 6(c)) **REQ-AN** surpasses the performance of all other evaluated analyses, deeming almost all task sets schedulable, even with a utilization of 90%. Figure 6(d) evaluates a larger range for the generated periods ($T_{min} = 10,\ T_{max} = 1000$), showing similar trends to those in Figure 6(a). Figures 6(e)–(g) report the schedulability achieved with larger task sets composed of $n = 15$ tasks, with (f)–(g) evaluating a larger range for the generated periods ($T_{min} = 10,\ T_{max} = 1000$). While (e) and (f) show overall similar trends as (a)–(d), when choosing consistently very long suspensions ($\beta_{min} = \beta_{max} = 0.6$) the schedulability of **REQ-AN** becomes slightly worse than **RTA-A** and **RTA-G** (Figure 6(g)), meaning that **REQ-AN** does not analytically dominate the existing techniques.

Constrained Deadlines: Experimental results for the case of constrained deadlines (with $\alpha = 0.8$) are provided in Figures 6(i)-(l), considering $n = 10$ or $n = 50$ tasks and periods in the range $[T_{min}{=}100,\ T_{max}{=}1000]$. We note that **RTA-G** was not evaluated since it is limited to implicit-deadline tasks. Similar trends as for the implicit-deadline cases can be observed, with **REQ-AN** surpassing all other analyses for EDF.

Analysis Runtimes: While the worst-case time complexity of our approach is discussed in Section III-F, Figures 7(a)–(b) report, respectively, the average and maximum analysis runtimes observed when running the experiment in Figure 6(e) on a machine equipped with an Intel Core i9-9900 processor and 128 GB of main memory. The average analysis runtimes measured for **REQ-AN** are overall similar to those of **DM**. The runtimes of **REQ-AN** are slightly longer at the utilization levels at which the schedulability curve in Figure 6(e) declines, i.e., where determining schedulability is more challenging. Furthermore, Figures 7(c)–(d) report the average and maximum runtimes observed when running the experiment in Figure 6(h), where the number of tasks n is increased to 50. We observe that in all evaluated configurations of Figure 7, the observed runtime for **REQ-AN** remains quite low. That is, our schedulability test for one task set runs less than 0.1 seconds to complete on average and less than 1 second to complete in the worst case.

Overall: Our experiments demonstrate that the proposed requirements-based schedulability test **REQ-AN** outperforms existing analyses for EDF scheduling by a significant margin in many cases, and performs only slightly worse than **RTA-A** and **RTA-G** for specific configurations. Furthermore, while the previous analyses for EDF consistently perform worse than **DM**, **REQ-AN** often reaches or even surpasses the performance of **DM**. Our analysis achieves all that with a runtime comparable to that of **DM**.

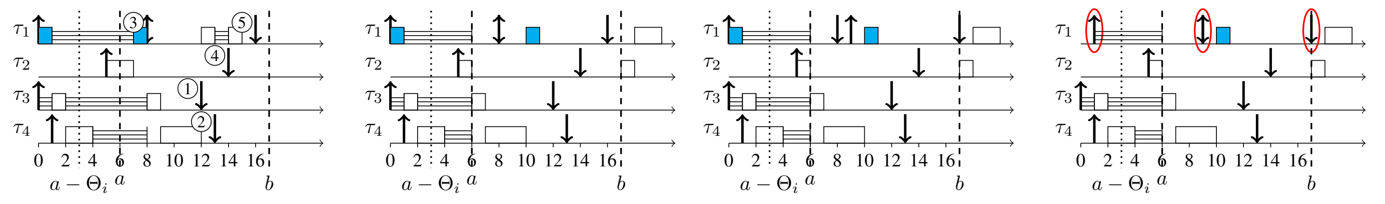
VII. Conclusion

We introduce a novel requirement-based schedulability analysis for dynamic self-suspending tasks under preemptive EDF. Our approach formalizes execution-exceedance requirements and iteratively tests and substitutes requirements by enlarging the analysis interval until a schedulability decision is reached. In particular, this method is the first to achieve a dynamic interval extension procedure, surpassing the limitations of existing fixed-interval analyses. Our evaluation demonstrates that requirement-based analysis outperforms state-of-the-art analysis by an often times large margin. Furthermore, it is the first EDF analysis that surpasses the results for DM found in the literature, marking a milestone in the analysis of EDF scheduling for self-suspending tasks.

Future work will extend our schedulability test further by exploring more sophisticated approaches to choose the threshold parameters Θ and by considering multiprocessor scheduling algorithms. Moreover, while our work focuses on discrete time to simplify the technical arguments for the proof of Lemma 7, we will investigate how to extend the results to the continuous time domain.

Appendix A
Proof of Lemma 7

Given a task set $\mathbb{T}$ such that Equation (11) does not hold, we define $W = \left\{\tau_i(j) \,\middle|\, r^{\omega}_{i,j} < a - \Theta_i \text{ and } r^{\omega}_{i,j} \text{ pending at } a\right\}$



(a) Schedule: $(L, E) = (11, 7)$ (b) After Step 1 (c) Step 2: Move releases after a (d) After Step 3

Figure 8. Illustration of the constructive proof for Lemma 7 using four tasks with implicit deadlines and $\Theta_i = 3$ for all i. Priority of jobs is indicated by small numbers (1 = highest priority). We have $W = \{\tau_1(1), \tau_3(1), \tau_4(1)\}$ and $\tau_1(1)$ (marked blue) has lowest priority in W, i.e., $\tau_1(1) = \tau_{i_{cr}}(j_{cr})$. A schedule with critical task $\tau_{i_{cr}} = \tau_1$ is constructed.

as the jobs with carry-in that are executed work-conservingly. First, we prove a helper lemma to show that $W \neq \emptyset$. Afterwards, we construct a system evolution ω and schedule $\mathcal{S} \in \sigma_\omega$ that satisfy conditions 1)–3) from Lemma 7 by choosing a task of W and making it periodic without reducing the workload from carry-in jobs.

Lemma 12. *Given ω, $\mathcal{S}$ and $I = [a, b]$ such that **C1*–C3*** and **EC** are satisfied, if Equation* (11) *does not hold, then $W \neq \emptyset$.*

Proof: We prove $W \neq \emptyset$ by contradiction. To that end, we assume that $W = \emptyset$. If **C1*–C3*** and **EC** are satisfied, we know that the amount of execution after a of jobs with deadline $\leq b$ exceeds E. Since $W = \emptyset$, no jobs $\tau_i(j)$ with deadline $\leq b$ and released before $a - \Theta_i$ are pending at a. Therefore, the total amount of execution time after a by jobs with deadline $\leq b$ is upper bounded by $\sum_{i=1}^{n} \left\lfloor \frac{L+T_i-D_i}{T_i} \right\rfloor C_i + \sum_{i \in \mathbb{I}^*(L,\Theta)} C_i$ and exceeds E. However, since Equation (11) does not hold, we reach a contradiction. ■

Since $\mathcal{T}^*_\Theta(L, E) = \mathit{True}$ by assumption of Lemma 7, we know that there exists ω, $\mathcal{S}$ and $I = [a, b]$ of length L such that **C1*–C3*** and **EC** are satisfied. Then, $W \neq \emptyset$ because of Lemma 12. We choose $\tau_{i_{cr}}(j_{cr})$ to be the lowest-priority job in W. An exemplary setting is depicted in Figure 8a with $\tau_{i_{cr}}(j_{cr}) = \tau_1(1)$ marked blue. Here, task thresholds are all set to $\Theta_i = 3$ for convenience in presentation. Please note that due to **C1***, the job prioritization does not have to follow EDF. We transform the schedule in three steps:

Step 1: Exploit work-conserving properties. For all jobs which can be executed non-work-conservingly (i.e., jobs of τ_i with release $\geq a - \Theta_i$) we move the execution time fully after b or the end of the carry-in workload by jobs of W. Further, we remove any suspension at or after a and we remove all jobs with deadline $\leq a$ which have lower priority than $\tau_{i_{cr}}(j_{cr})$[5]. This step is depicted in Figure 8b. Please note that the jobs in W cannot be executed non-work-conservingly. After this conversion, conditions **C1*–C3*** are still satisfied. Furthermore, since the schedule of carry-in jobs in W before a remains unchanged, they still provide the same amount of carry-in and **EC** is satisfied.

Step 2: Move job releases after a. For each task τ_i, all jobs with release and deadline in $I = [a, b]$ are moved backwards such that the latest job releases at $b - D_i$, the second-latest job releases at $b - D_i - T_i$ and the x-th latest at $b - D_i - (x-1)T_i$. This transformation is depicted in Figure 8c. In the illustrated example, only the job $\tau_1(2)$ is released and has deadline during I. Its release is moved such that its deadline aligns with b. Please note that this step does not change the schedule $\mathcal{S}$ because all execution of the moved jobs is already moved to the right by step 1. Rather, it only modifies the release times specified in ω. In particular, **C1*–C3*** and **EC** remain satisfied.

Step 3: Move job release of $\tau_{i_{cr}}(j_{cr})$. We postpone the release time $r^\omega_{i_{cr},j_{cr}}$ of $\tau_{i_{cr}}(j_{cr})$ incrementally—one time unit at a time. That is, we remove any execution or suspension that has occured for $\tau_{i_{cr}}(j_{cr})$ at $r^\omega_{i_{cr},j_{cr}}$ and replace $r^\omega_{i_{cr},j_{cr}}$ by $r^\omega_{i_{cr},j_{cr}} + 1$. This modification is depicted in Figure 8d, where the release of $\tau_1(1)$ is moved from 0 to 1, and one execution unit is removed from the execution-suspension-pattern of $\tau_1(1)$. This transformation has the following impact:

- The transformation has *no impact* on all jobs with higher priority than $\tau_{i_{cr}}(j_{cr})$. This includes all jobs in $W \setminus \{\tau_{i_{cr}}(j_{cr})\}$ since $\tau_{i_{cr}}(j_{cr})$ has the lowest priority of W by definition.
- The transformation might change the schedule of lower-priority jobs. However, lower-priority jobs are not part of W since $\tau_{i_{cr}}(j_{cr})$ has the lowest priority of W. Hence, if lower-priority jobs are executed after a, they are executed non-work-conservingly (with execution moved after b or after the carry-in by Step 1). Therefore, the execution of such jobs which occurs after a remains after a.

Due to these properties, the accumulated execution time after a remains $> E$, i.e., **EC** remains satisfied. Furthermore, **C1*** and **C2*** are unaffected by this step. Since there are no jobs with lower priority than $\tau_{i_{cr}}(j_{cr})$ and deadline $\leq a$ in the system (due to Step 1), also **C3*** is satisfied. We iteratively move forward the release of $\tau_{i_{cr}}(j_{cr})$, safely preserving conditions **C1*–C3*** and **EC**, until either (A) $r^\omega_{i_{cr},j_{cr}} = b - \left\lceil \frac{L+T_{i_{cr}}-D_{i_{cr}}}{T_{i_{cr}}} \right\rceil \cdot T_{i_{cr}} + T_{i_{cr}} - D_{i_{cr}}$ or (B) $r^\omega_{i_{cr},j_{cr}} \geq a - \Theta_i$. In the former case (A), the proof is completed. In the latter case (B), $\tau_{i_{cr}}(j_{cr})$ is not in the set W anymore, i.e., the number of jobs in W is reduced by 1. We can continue choosing the new lowest-priority job as $\tau_{i_{cr}}(j_{cr})$ and starting the transformation again from Step 1. This process will terminate with case (A) at some point because we have shown in Lemma 12 that W cannot become empty. ■

[5]This might affect the interference on even lower priority carry-in jobs which are not in W. However, since their execution after a is executed non-work-conservingly, their interference after a cannot decrease.

Acknowledgment

This result is part of a project (PropRT) that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 865170). This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

[1] F. Aromolo, A. Biondi, and G. Nelissen. Response-time analysis for self-suspending tasks under EDF scheduling. In *34th Euromicro Conference on Real-Time Systems, ECRTS*, pages 13:1–13:18, 2022.

[2] F. Aromolo, A. Biondi, G. Nelissen, and G. Buttazzo. Event-driven delay-induced tasks: Model, analysis, and applications. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 53–65. IEEE, 2021.

[3] 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.

[4] S. Baruah. Techniques for multiprocessor global schedulability analysis. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, pages 119–128, 2007.

[5] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *proceedings Real-Time Systems Symposium (RTSS)*, pages 182–190, Dec 1990.

[6] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[7] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo. A framework for supporting real-time applications on dynamic reconfigurable fpgas. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 1–12. IEEE, 2016.

[8] B. B. Brandenburg. Multiprocessor real-time locking protocols. In *Handbook of Real-Time Computing*, pages 347–446. Springer, 2022.

[9] J.-J. Chen, G. Nelissen, and W.-H. Huang. A unifying response time analysis framework for dynamic self-suspending tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 61–71, 2016.

[10] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, N. C. Audsley, R. Rajkumar, D. de Niz, and G. von der Brüggen. Many suspensions, many problems: a review of self-suspending tasks in real-time systems. *Real Time Syst.*, 55(1):144–207, 2019.

[11] U. C. Devi. An improved schedulability test for uniprocessor periodic task systems. In *15th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 23–32, 2003.

[12] Z. Dong and C. Liu. Closing the loop for the selective conversion approach: A utilization-based test for hard real-time suspending task systems. In *RTSS*, pages 339–350. IEEE Computer Society, 2016.

[13] M. Guenzel, M. Sudvarg, M. Deppert, A. Li, N. Zhang, and J.-J. Chen. Optimal priority assignment for synchronous harmonic tasks with dynamic self-suspension. In *31st IEEE Real-Time and Embedded Technology and Applications Symposium*, 2025.

[14] M. Günzel, F. Aromolo, A. Biondi, and J.-J. Chen. Requirement-based analysis of self-suspending tasks under EDF (artifact), 2025. https://doi.org/10.5281/zenodo.17203679. Zenodo.

[15] M. Günzel and J. Chen. Correspondence article: Counterexample for suspension-aware schedulability analysis of EDF scheduling. *Real Time Syst.*, 56(4):490–493, 2020.

[16] M. Günzel, G. von der Brüggen, and J.-J. Chen. Suspension-aware earliest-deadline-first scheduling analysis. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 39(11):4205–4216, 2020.

[17] M. Günzel, G. von der Brüggen, and J.-J. Chen. Tighter worst-case response time bounds for jitter-based self-suspension analysis. In *36th Euromicro Conference on Real-Time Systems, ECRTS*, volume 298, pages 4:1–4:24, 2024.

[18] M. Günzel, G. von der Brüggen, K.-H. Chen, and J.-J. Chen. EDF-like scheduling for self-suspending real-time tasks. In *IEEE Real-Time Systems Symposium, (RTSS)*, pages 172–184, 2022.

[19] 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 (DAC)*, pages 154:1–154:6, 2015.

[20] J. Lee. Improved schedulability analysis using carry-in limitation for non-preemptive fixed-priority multiprocessor scheduling. *IEEE Trans. Computers*, 66(10):1816–1823, 2017.

[21] C. Liu and J. H. Anderson. Suspension-aware analysis for hard real-time multiprocessor scheduling. In *25th Euromicro Conference on Real-Time Systems, ECRTS*, pages 271–281, 2013.

[22] 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.

[23] 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, 1973.

[24] 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.

[25] M. Spuri. Analysis of deadline scheduled real-time systems. Research Report RR-2772, INRIA, France, 1996.

[26] Y. Wang, B. Lv, Q. Zhou, J. Li, and T. Tan. Schedulability analysis for self-suspending tasks under edf-like scheduling. *IEEE Transactions on Computers*, 2025.