Suspension-Aware Earliest-Deadline-First Scheduling Analysis

Mario Günzel, *Graduate Student Member, IEEE*, Georg von der Brüggen, *Member, IEEE*, and Jian-Jia Chen, *Senior Member, IEEE*

***Abstract*—While the earliest-deadline-first (EDF) scheduling algorithm has extensively been utilized in real-time systems, there is almost no literature considering EDF for task sets with dynamic self-suspension behavior. To be precise, there is no specialized result for uniprocessor systems, besides the trivial suspension-oblivious approach. The work by Liu and Anderson (in ECRTS 2013) and Dong and Liu (in RTSS 2016) for suspension-aware multiprocessor global EDF can also be applied to uniprocessor systems and therefore be considered the state-of-the-art. In this work, two novel schedulability analyses (one for sporadic and one for periodic task sets) for suspension-aware EDF on uniprocessor systems are proposed, which outperform the state-of-the-art on such systems in empirical and theoretical comparison. We further show that the analysis by Dong and Liu is in fact not suspension-aware for uniprocessor systems.**

***Index Terms*—Embedded software, real time systems.**

I. Introduction

MANY embedded systems are comprised of recurrently executed tasks that have to comply to timing constraints, e.g., a relative deadline related to the release time of a task instance. Such real-time systems require both functional and timing correctness and therefore (deterministic or probabilistic) timing guarantees must be provided. This is especially complex when a computation task assigned to the processor may suspend itself, i.e., release the processor despite being incomplete, to wait, for instance, for hardware accelerators (such as GPUs) or shared resources managed by a multiprocessor locking protocol [9, Sec. 2]. One reason is that most important classical timing analysis concepts, e.g., the critical instant theorem for static-priority scheduling by Liu and Layland [16] and the demand bound function for earliest-deadline-first (EDF) scheduling by Baruah *et al.* [1], assume a work-conserving task behavior and are hence invalid in the presence of self-suspension. Due to the high problem complexity, a large number of research results in the area of self-suspension have, unfortunately, recently been reported flawed, as detailed in [9] where the authors listed six categories of flaws and over 20 affected publications.

Applications of self-suspension in real-time embedded systems are for example, I/O- or memory-intensive tasks, multiprocessor synchronization, hardware acceleration by using co-processors and computation offloading, etc., detailed in [9, Sec. 2] Two specific self-suspension task models have been extensively explored in the literature, namely, the *segmented* (or *multisegment*) self-suspension and the *dynamic* self-suspension model. In the segmented self-suspension model all jobs of a task have the same execution behavior, which is predefined by a sequence of computation segments and self-suspension intervals. In the dynamic self-suspension task model, each task is defined by its *total* worst-case execution time and its *total* maximum self-suspension time, where a job of task τ_i can exhibit any number of self-suspensions of arbitrary duration as long as the sum of the suspension (respectively, execution) interval lengths does not exceed the specified maximum self-suspension (respectively, worst-case execution) time.

The segmented self-suspension model is suitable when the execution of each instance of a task always follows a specific structure. Prime examples are situations where a significant part of the computation is offloaded to hardware accelerators, which often can be modeled as a three-step process: 1) local computation segment preparing the offloaded data, e.g., data compression; 2) computation on the accelerator, i.e., the time the task instance is suspended; and 3) a local computation segment for post processing.

However, such a clear structure of a task's self-suspension behavior cannot always be assumed. For instance, it has been reported that suspension-aware analysis can be applied to handle suspension-based locks for multiprocessor synchronization due to lock contention, usually termed as *remote blocking*. Specifically, when partitioned or semi-partitioned multiprocessor scheduling paradigms are applied, the problem can be modeled as self-suspending tasks in uniprocessor systems. For details, please refer to [4] and [9]. Depending on different execution paths/conditions, a task may access different mutually exclusive shared resources, and suspend when waiting for remote resource access. Therefore, for a specific task, the access pattern to the shared resources may differ for different task instances, i.e., the number of resource accesses, their order, their location, and which resources are accessed changes for individual task instances. As a result, the dynamic self-suspension model is often utilized when considering multiprocessor resource sharing protocols. Therefore, all the

Manuscript received April 17, 2020; revised June 17, 2020; accepted July 6, 2020. Date of publication October 2, 2020; date of current version October 27, 2020. This work was supported in part by the Deutsche Forschungsgemeinschaft Project Sus-Aware under Project 398602212, and in part by SFB876-A1 under Project 124020371. This article was presented in the International Conference on Embedded Software 2020 and appears as part of the ESWEEK-TCAD special issue. *(Corresponding author: Mario Günzel.)*

The authors are with the Department of Informatics, TU Dortmund University, 44227 Dortmund, Germany (e-mail: mario.guenzel@tu-dortmund.de; georg.von-der-brueggen@tu-dortmund.de; jian-jia.chen@cs.tu-dortmund.de).

Digital Object Identifier 10.1109/TCAD.2020.3013095

results reported in [9, Sec. 6] applied dynamic self-suspension model to handle such dynamics.

While the segmented model can be seen as a precise but restrictive model, the dynamic model is flexible but imprecise when there is a small set of fixed suspension patterns. Hence, hybrid self-suspension models that bridge the gap between these two models have been recently proposed by von der Brüggen *et al.* [22]. Which of these models is applied should be decided based on the available information. Therefore, scheduling algorithms and their schedulability tests for each of the three models are required for different scenarios.

The well-defined suspension structure in the segmented self-suspension model can be exploited to optimize the scheduling policies, e.g., the fixed-relative-deadline (FRD) strategies in [7], [19], and [21] and mapping to the master-slave problem in [6]. Chen [5] and Mohaqeqi *et al.* [17] showed that verifying whether a set of segmented self-suspension tasks can meet their deadlines under static-priority scheduling is co$\mathcal{NP}$-hard in the strong sense.

For the dynamic self-suspension model the computational complexity of scheduler design remains an open problem [5], [10]. Under static-priority scheduling, Chen *et al.* [8] provided a unifying response time analysis (RTA) framework, considering multiple approaches to model the self-suspension time under a dynamic self-suspension behavior, i.e., as computation, carry-in, blocking, or jitter.

For the dynamic self-suspension model under dynamic-priority scheduling, the only well-known algorithm that has been analyzed in the literature is preemptive EDF [16], which assigns priorities to jobs due to their absolute deadlines. Under EDF, jobs with an earlier absolute deadline have a higher priority than those with later deadlines and ties are broken arbitrarily. Although EDF is not optimal for scheduling self-suspending task systems as shown in [20], EDF remains one of the most adopted scheduling strategies.

The overall aim to create suspension-aware analysis is to avoid counting suspensions that happen at the same time. This can be achieved either by being pessimistic with counting workload and dropping suspension completely, as we do in Section IV, or by only removing the part of suspension which is redundant from the analysis, as in Section V. Besides the trivial suspension-oblivious schedulability analysis, the following results have been provided in the literature:

1) Analysis by Devi [11, Th. 8], dating back to 2003, dedicated to uniprocessor EDF. The review paper in [9] notes that Devi does not give a proof of her analysis. This analysis was proven to be wrong by a counterexample provided by Günzel and Chen [14].
2) Analysis by Liu and Anderson [15], for global multiprocessor EDF. The analysis is valid for uniprocessor EDF by setting the number of processors to 1.
3) Analysis by Dong and Liu [12], for global multiprocessor EDF. The analysis is valid for uniprocessor EDF by setting the number of processors to 1.

Despite the possibility to apply the analyses of Liu and Anderson [15] and Dong and Liu [12], the fundamental research question regarding the schedulability tests for uniprocessor EDF considering dynamic self-suspension remains open. In particular, there is no schedulability analysis that is superior to the trivial suspension-oblivious analysis for uniprocessor EDF. We believe that the fundamental knowledge of uniprocessor EDF is a cornerstone to achieve tight schedulability tests for more advanced multiprocessor scenarios.

Contributions: In this article, we focus on schedulability analysis for the dynamic self-suspension model under preemptive EDF on a uniprocessor and provide two sufficient schedulability tests for this setting. We limit our attention to *implicit-deadline* real-time task systems, i.e., the relative deadline of a task is the same as its period (for periodic releases) or minimum interarrival time (for sporadic releases). We prove that the utilization-based schedulability test by Dong and Liu [12] does not improve the suspension-oblivious approach, if it is utilized for uniprocessor systems, in Section III. Our contributions are as follows.

1) We provide an RTA for EDF in Section IV. If all tasks have suspension, then this test dominates the test developed by Liu and Anderson [15] applied to implicit-deadline *sporadic* real-time tasks in uniprocessor systems. We also, present how this analysis can be easily extended to multiprocessor global EDF in Section IV-C.
2) We further provide a utilization-based schedulability test for *periodic* task systems in Section V, where we estimate the amount of time where the processor is used by other jobs during self-suspension. This is the *first* schedulability test which dominates and improves the trivial suspension-oblivious approach for uniprocessor systems.
3) In addition to the theoretical improvement and dominance of existing methods under different conditions, the evaluation results in Section VI show that the response-time-aware schedulability test performs significantly better than the state-of-the-art.

II. Task Model, System Model, and Notation

We consider a set $\mathbb{T} = \{\tau_1, \ldots, \tau_k, \ldots, \tau_n\}$ of n independent recurrent real-time tasks, described by the dynamic self-suspension model, in a uniprocessor system. Each task τ_i releases an infinite number of task instances, called jobs, and is described by $\tau_i = (C_i, S_i, D_i, T_i)$. The jobs are released *sporadically* according to the minimum interarrival time (or period) $T_i > 0$, which means that two consecutive job releases are separated by at least T_i, and have a relative deadline $D_i > 0$. We assume an *implicit-deadline* task set, i.e., $T_i = D_i$ for all tasks $\tau_i \in \mathbb{T}$. The execution and suspension behavior of a task is described by its *worst-case execution time* $C_i \in [0, D_i]$ and its *maximum suspension time* $S_i \in [0, D_i]$. A task set is called *schedulable*, if for each task τ_i each job $\tau_{i,j}$ (with $j \in \mathbb{N}$) meets its deadline, i.e., a job $\tau_{i,j}$ released at time $r_{i,j}$ must be able to execute C_i time units before its absolute deadline $d_{i,j} = r_{i,j} + D_i = r_{i,j} + T_i$. In addition, according to the *dynamic* self-suspension model, each job may suspend itself as often as desired as long as the maximum suspension time constraint is not violated, where suspension means that the job that is

currently executed relinquishes the processor. While this may allow other jobs to be executed, it may also result in the processor running idle. For each task $\tau_i \in \mathbb{T}$, the *utilization* is defined as $U_i = (C_i/T_i)$, and the *total utilization* of the task set is $U_{\text{sum}} = \sum_{i=1}^{n} U_i$. We assume that $C_i + S_i \leq D_i$ for all tasks $\tau_i \in \mathbb{T}$, since otherwise τ_i is not schedulable by default.

A special case of sporadic real-time systems are *periodic* real-time systems, in which a periodic task τ_i releases its jobs periodically with *period* T_i. For a periodic task $\tau_i = (C_i, S_i, D_i, T_i, \phi_i)$, in addition to the previously defined parameters, the first job release of task τ_i happens at time ϕ_i and is assumed to happen at time 0 if ϕ_i is omitted.

In this work, we examine how the worst-case response time of a specific job J of task τ_k can be calculated when $\mathbb{T}$ is scheduled by preemptive EDF scheduling [16], where jobs with a smaller deadline have higher priority and ties are broken arbitrarily. The analysis in Section IV is dedicated to sporadic task systems, whilst the one in Section V is only valid for periodic task systems. Both analyses assume dense time.

We define the following terminology to describe the possible states of the processor P, both in relation to J.

Definition 1: At time t, the processor P is:

1) *working on* J, when J is the job executed by P, i.e., the remaining execution time of J is reduced;
2) *suspended for* J, when J has suspended itself, i.e., the remaining suspension time of J is reduced.

Definition 2: At time t, the processor P is:

1) *working*, while it is working on any job;
2) *suspended*, while it is suspended for at least one job but not working on any job, i.e., the processor idles but at least one previously released job has not been finished.
3) *waiting*, while it is neither working nor suspended, i.e., the processor idles and all previously released jobs have been finished.

Remark 1: This distinction is made for the following reasons.

1) If the processor is working on a job, then it is unavailable for all lower priority jobs.
2) If the processor is suspended for a job, it is available to lower priority jobs.

III. Existing Methods

In this section, we recap existing schedulability tests for dynamic self-suspending tasks under preemptive EDF scheduling [16] that are applicable to implicit-deadline uniprocessor systems. The method by Devi [11, Th. 8] is left out, since a counterexample for the analysis has been provided in [14].

A. Suspension-Oblivious

Suspension-oblivious analysis interprets suspension time as additional computation time. This approach can be very pessimistic, especially if some tasks have a large maximum suspension time. Since implicit-deadline task sets without suspension are schedulable under EDF if and only if their total utilization is less or equal to 1, the schedulability test (detailed in [9, Sec. 4]) is defined as follows.

Theorem 1 (Suspension-Oblivious): Let $\mathbb{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a system of n implicit-deadline sporadic tasks with dynamic self-suspension. Then $\mathbb{T}$ is schedulable using preemptive EDF if $\sum_{i=1}^{n} [(C_i + S_i)/T_i] \leq 1$.

B. Workload-Based Schedulability Test

The method by Liu and Anderson [15] is actually formulated for multiprocessor systems, considering arbitrary-deadline sporadic tasks.[1] Furthermore, they added a *tardiness threshold*, which is an upper bound on the amount of time by which individual jobs may miss their deadline, to make their analysis also available for soft real-time systems (where a deadline overrun is considered a service degradation but not system failure). Setting the number of processors to 1 and the tardiness threshold to 0, their method can be applied to implicit-deadline sporadic tasks in uniprocessor systems. For each task $\tau_l \in \mathbb{T}$ they estimate the workload W_c and W_{nc} with and without carry-in jobs[2] inside an interval of length ξ_l and check whether the processor still has enough capacity to work on and be suspended for a job of τ_l.[3]

Theorem 2 (Liu and Anderson [15]): Let $\mathbb{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a system of n implicit-deadline sporadic tasks with dynamic self-suspension. The task set is schedulable using preemptive EDF if for all $\tau_l \in \mathbb{T}$, for all $s_{l,j} \in \{0, 1, \ldots, S_l\}$, and for all $\xi_l \in [T_l, [(C_l + s_{l,j} + \sum_{i=1}^{n} C_i)/((1 - \sum_{i=1}^{n}(C_i/T_i)))]) \cap \mathbb{Z}$ the property

$$\sum_{\tau_i \in \tau^s} \max\{W_{nc}(\tau_i), W_c(\tau_i)\} + \sum_{\tau_j \in \tau^e} W_{nc}(\tau_j) \leq \xi_l - C_l - s_{l,j} \quad (1)$$

holds, where τ^s and τ^e are subsets of $\mathbb{T}$ consisting of those tasks τ_i which do have suspension time ($S_i \neq 0$) and those which do not have suspension ($S_i = 0$), i.e., which are *execution-only*, and the values $W_c(\tau_i)$ and $W_{nc}(\tau_i)$ are estimations for relevant work of τ_i with and without carry-in jobs, defined by following.

1) $W_c(\tau_i) = \begin{cases} \min\{\Delta(\tau_i, \xi_l), \xi_l - C_l - s_{l,j} + 1\}, & i \neq l \\ \min\{\Delta(\tau_l, \xi_l) - C_l, \xi_l - T_l\}, & i = l \end{cases}$
with $\Delta(\tau_i, t) = (\lceil (t/T_i) \rceil - 1)C_i + \min\{C_i, t - \lceil (t/T_i) \rceil T_i + T_i\}$.
2) $W_{nc}(\tau_i) = \begin{cases} \min\{\lfloor (\xi_l/T_i) \rfloor C_i, \xi_l - C_l - s_{l,j} + 1\}, & i \neq l \\ \min\{\lfloor (\xi_l/T_l) \rfloor C_l - C_l, \xi_l - T_l\}, & i = l. \end{cases}$

We remark that Liu and Anderson [15] restrict only to discrete time for their analysis, i.e., the periods, worst-case execution times, and maximum suspension times of all tasks are in $\mathbb{Z}_{\geq 0}$. This is not necessary for the methods which we provide in Sections IV and V.

C. Utilization-Based Schedulability Test

The method by Dong and Liu [12, Th. 2] is also formulated for multiprocessor systems. By setting the number of processors to 1 their method can be utilized for the dynamic self-suspension uniprocessor task model.

[1] For *arbitrary-deadline* tasks, there is no constraint on the relation of deadline and period.

[2] Carry-in jobs are those which are released before the interval under analysis and still interfere during the interval.

[3] We note that there is a typing error in this article, corrected here.

Theorem 3 (Dong and Liu [12, Th. 2]): Let $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$ be a system of n implicit-deadline sporadic tasks with dynamic self-suspension. The task set is schedulable using preemptive EDF scheduling if $\sum_{i=1}^{n} [(C_i + S_i)/T_i] \leq 1$ holds or if there exists some $\mathbb{T}_0 \subseteq \mathbb{T}$ and some $k \in \{2, \ldots, |\mathbb{T}_0|\}$ with $\sum_{i=1}^{\lfloor (k/2) \rfloor} E^i(\mathbb{T}_0) \geq T_{\max}(\mathbb{T}_0)$, where $E^i(\mathbb{T}_0)$ is the ith minimum $(C_j + S_j)$ among tasks $\tau_j \in \mathbb{T}_0$ and $T_{\max}(\mathbb{T}_0)$ is the maximum task period of tasks in $\mathbb{T}_0$, such that

$$\sum_{\tau_i \in \mathbb{T}} \frac{C_i}{T_i} + \sum_{\tau_i \in \mathbb{T} - \mathbb{T}_0} \frac{S_i}{T_i} + \sum_{i=1}^{k} v^i(\mathbb{T}_0) \leq 1 \tag{2}$$

where $v^i(\mathbb{T}_0)$ is the ith maximum suspension ratio (S_j/T_j) among tasks $\tau_j \in \mathbb{T}_0$.

In fact this method is no improvement compared to the suspension-oblivious schedulability test if it is formulated for only one processor.

Proposition 1: The schedulability test by Dong and Liu as formulated in Theorem 3 for uniprocessor systems is identical to the suspension-oblivious test as in Theorem 1.

Proof: Assume that the schedulability test from Theorem 3 is superior to the suspension oblivious approach for the task set $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$. Then, we find some $\mathbb{T}_0 \subseteq \mathbb{T}$ and some $k \in \{2, \ldots, |\mathbb{T}_0|\}$ with $\sum_{i=1}^{\lfloor (k/2) \rfloor} E^i(\mathbb{T}_0) \geq T_{\max}(\mathbb{T}_0)$. By reordering we assume that $\mathbb{T}_0 = \{\tau_1, \ldots, \tau_{n_0}\}$ and $E^i(\mathbb{T}_0) = C_i + S_i$. Then we have

$$\begin{aligned} 1 &\leq \sum_{i=1}^{\left\lfloor \frac{k}{2} \right\rfloor} \frac{E^i(\mathbb{T}_0)}{T_{\max}(\mathbb{T}_0)} < \sum_{i=1}^{k} \frac{E^i(\mathbb{T}_0)}{T_{\max}(\mathbb{T}_0)} \leq \sum_{i=1}^{k} \frac{C_i + S_i}{T_i} \\ &\leq \sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{k} v^i(\mathbb{T}_0) + \sum_{i=n_0+1}^{n} \frac{S_i}{T_i}. \end{aligned}$$

In other words, $\sum_{i=1}^{\lfloor (k/2) \rfloor} E^i(\mathbb{T}_0) \geq T_{\max}(\mathbb{T}_0)$ implies that (2) does not hold. Therefore, the schedulability test in Theorem 3 coincides with the suspension-oblivious schedulability test. We note that in the proof we utilized that $E^{\lfloor (k/2) \rfloor + 1}(\mathbb{T}_0) > 0$, but this holds since $0 < T_{\max}(\mathbb{T}_0) \leq \sum_{i=1}^{\lfloor (k/2) \rfloor} E^i(\mathbb{T}_0) \leq \lfloor (k/2) \rfloor \cdot E^{\lfloor (k/2) \rfloor}(\mathbb{T}_0) \leq \lfloor (k/2) \rfloor \cdot E^{\lfloor (k/2) \rfloor + 1}(\mathbb{T}_0)$. ∎

In the remaining part of the paper, we omit this schedulability test in the discussions since it is the same as the suspension-oblivious approach for uniprocessor systems.

IV. Method 1: Response Time Analysis

In this section, we introduce a new method to test the schedulability of an implicit-deadline sporadic task set under preemptive EDF scheduling. For this purpose, we estimate the interference from higher-priority jobs to calculate an upper bound on the worst-case response time for each task. Although RTA has been conducted for EDF in the classical setting without self-suspension [2], [18], rigorous proofs are required to derive suspension-aware RTA since it has been reported that some seemingly correct extensions from classical real-time schedulability analyses can be flawed, as explained in [9].

Let $\tau_k \in \mathbb{T} = \{\tau_1, \ldots, \tau_n\}$ be any task and J some job of τ_k. We denote the release time and the deadline of J by r_J and $d_J = r_J + T_k$, respectively. Let $B_J^i(t)$ be the interference by higher priority jobs of τ_i. The time where the processor is working on jobs with higher priority than J, i.e., where it is unavailable to J, during an interval $[r_J, r_J + t] \subseteq [r_J, d_J]$ is $\sum_{i=1}^{n} B_J^i(t)$. The processor can work on and be suspended for J during an interval $[r_J, r_J + t]$ for $t - \sum_{i=1}^{n} B_J^i(t)$ time units. Hence, it finishes J during this interval if $C_k + S_k \leq t - \sum_{i=1}^{n} B_J^i(t)$. Equivalently, if J can not be finished during $[r_J, r_J + t]$, then $C_k + S_k > t - \sum_{i=1}^{n} B_J^i(t)$.

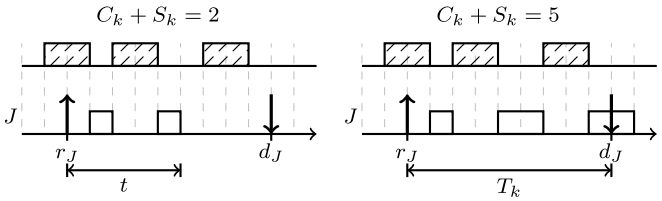

Fig. 1. Illustration of observations 1 (left) and 2 (right). Patterned areas are interference by higher priority jobs.

We derive the following two observations as depicted in Fig. 1, which provide a general upper bound on the response time and a sufficient schedulability condition. These are (mainly implicitly) used in a similar way in classical RTA and the underlying concepts are already applied for self-suspending tasks as in [15].

Observation 1: If J meets its deadline, then we calculate an upper bound on the response time of J by choosing the minimal t with $0 \leq t \leq T_k$, such that

$$C_k + S_k + \sum_{i=1}^{n} B_J^i(t) \leq t. \tag{3}$$

If no such t exists, then we can not give a guaranteed worst-case response time, but the job may still meet its deadline, e.g., if the job completes earlier than in its worst case.

Observation 2: If J does not meet its deadline, then it can not be finished during the interval $[r_k, r_k + T_k]$, i.e.,

$$C_k + S_k + \sum_{i=1}^{n} B_J^i(T_k) > T_k \tag{4}$$

as illustrated on the right hand side of Fig. 2.

We start by determining the higher priority jobs which may contribute to the response time of J. By the definition of EDF scheduling, jobs with a larger absolute deadline than J have a lower priority. We remove these jobs from the schedule for ease of notation, since they do not affect the response time of J. All jobs of the remaining schedule have deadline at most d_J and therefore may have a higher priority than J.

When determining which higher priority jobs contribute to the response time of J, we assume that all higher priority jobs meet their deadline. Furthermore, for simplicity, we assume that J is released at time $r_J = 0$. Other job releases and task phases are shifted accordingly (to negative time instants if necessary) and in the following, we consider the schedule after this shift.

We know that $B_J^k(t) = 0$ since all jobs of τ_k with higher priority than $J \in \tau_k$ meet their deadline which is at most 0. For a task τ_i with $i \neq k$ there are at most $\lfloor (T_k/T_i) \rfloor$ of the

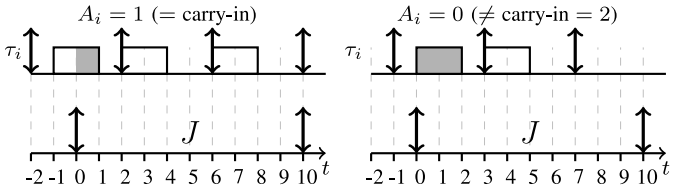

Fig. 2. Interference by higher priority jobs of task $\tau_i = (2, 0, 4, 4)$ during the interval $[r_J, f_J] = [0, 10]$. Left: interference = 5. Right: interference = 4.

remaining jobs released in $[0, t] \subseteq [0, T_k]$. Furthermore, there may be one additional overlapping job which is released before 0 but finishes after 0.

Lemma 1: For a task $\tau_i \neq \tau_k$ only the last $\lfloor (T_k/T_i) \rfloor + 1$ jobs[4] may be executed in $[0, t]$.

Proof: The other jobs of τ_i have a deadline of at most $T_k - (\lfloor (T_k/T_i) \rfloor + 1) \cdot T_i \leq T_k - (T_k/T_i) \cdot T_i = 0$. Since they meet their deadline, they are finished before the release of J. ■

Remark 2: In fact only the last $\lceil (T_k/T_i) \rceil$ jobs may be executed in $[0, t]$, which can be shown by a similar proof. Only for convenience, we use the bound given in the lemma and set the interference by the additional job to 0 if necessary.

For the last $\lfloor (T_k/T_i) \rfloor$ jobs of τ_i we estimate the time they may be executed in $[0, t]$ by C_i each. Although this seems to be very pessimistic if t is very small, this estimation is sufficient for our analysis. However, we have to be more careful with the $(\lfloor (T_k/T_i) \rfloor + 1)$th last job of τ_i which is released before 0 but may still finish after 0. Note that by our assumption, 0 is not a lower bound on the phase but the release of J.

Definition 3: We define J^i to be the $(\lfloor (T_k/T_i) \rfloor + 1)$th last job of τ_i. The time J^i is interfering in $[0, t]$ is denoted by A_i.

To derive a good response time bound in general, estimation of the carry-in, i.e., interference by higher priority jobs which are released before the job under analysis, is essential as outlined in [2]. Since A_i is the interference by a certain job released before J, it serves the role of the carry-in. Nevertheless, there is a slight difference as depicted in Fig. 2. Although in the scenario on the left-hand side (LHS) the carry-in and A_i coincide, the value of A_i is 0 in the right scenario since the third last job of τ_i finishes before J is released. We observe that A_i has its highest value when the interference is maximized, i.e., on the LHS. Hence, independently examining A_i is relatively safe for estimating interference. This is not the case for the carry-in, for which in general more caution is required.

With this definition of A_i, the time a task $\tau_i \neq \tau_k$ is interfering during $[0, t]$ is

$$B_J^i(t) \leq \left\lfloor \frac{T_k}{T_i} \right\rfloor \cdot C_i + A_i \tag{5}$$

and we will examine how to estimate the interference A_i by J^i during $[0, t]$ later on. First, we show how the estimation for $B_J^i(t)$ can be utilized, assuming that A_i is given.

[4]We highlight that we consider a limited schedule here, which was obtained by the original schedule. The *last x jobs* are the first x jobs counted from the end of the schedule.

Lemma 2: We consider a job J of τ_k. If all higher priority jobs meet their deadline and

$$S_k + C_k + \sum_{i \neq k} \left(\left\lfloor \frac{T_k}{T_i} \right\rfloor \cdot C_i + A_i \right) \leq T_k. \tag{6}$$

holds, then also J meets its deadline.

Proof: If J does not meet its deadline, then we obtain by Observation 2 that $T_k < S_k + C_k + \sum_{i \neq k} B_J^i(T_k) \leq S_k + C_k + \sum_{i \neq k} (\lfloor (T_k/T_i) \rfloor \cdot C_i + A_i)$, which contradicts (6). ■

Under the assumption, that A_i is given for all i, our schedulability test can be formulated as follows.

Proposition 2: Let $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$ be an implicit-deadline task set. If the equation

$$\widetilde{r}_k := S_k + C_k + \sum_{i \neq k} \left(\left\lfloor \frac{T_k}{T_i} \right\rfloor \cdot C_i + A_i \right) \leq T_k \tag{7}$$

holds for all indices $k \in \{1, \ldots, n\}$, then $\mathbb{T}$ is schedulable by preemptive EDF and the worst-case response time of τ_k is upper bounded by $\widetilde{r}_k$ for each $k \in \{1, \ldots, n\}$.

Proof: We use an indirect proof to show that $\mathbb{T}$ is schedulable by preemptive EDF. In the following, we assume that $\mathbb{T}$ is not schedulable. Let J be the job with the highest priority whose deadline is not met and let τ_k be the task corresponding to J. In this case, Lemma 2 shows that J is schedulable, which contradicts our assumption.

We have proven that $\mathbb{T}$ is schedulable and are able to use (5) together with (3) to upper bound the response time of each job of τ_k by $t = \widetilde{r}_k$ for $k = 1, \ldots, n$. ■

The aforementioned proposition analyses the scenario with worst-case release pattern, i.e., all jobs are released as late as possible and the interference is maximized as on the LHS of Fig. 2. In such a scenario A_i and the carry-in coincide. Hence, in the following estimation of A_i we observe mainly typical carry-in bounds as provided in [2], [15], and [18]. Hereinafter, we provide rigorous proofs where suspension is integrated seamlessly into carry-in analysis.

For the estimation of A_i we introduce two approaches, where the first one examines the deadline of the overlapping job J^i while the second one makes use of its response time.

Lemma 3: We can estimate A_i by

$$A_i \leq \min \left\{ C_i, T_k - \left\lfloor \frac{T_k}{T_i} \right\rfloor \cdot T_i \right\} =: \widetilde{A}_i^1. \tag{8}$$

Proof: Since there are $\lfloor (T_k/T_i) \rfloor$ other jobs of τ_i between the deadline of J^i and T_k, the deadline of J^i is at most $d_{J^i} = T_k - \lfloor (T_k/T_i) \rfloor \cdot T_i$. Due to the assumption that all jobs with higher-priority than J meet their deadline, J^i meets its deadline as well and any possible interference by J^i must happen in the interval from $[0, d_{J^i}]$.

On the other hand, we also know that the interference by J^i is bounded by the worst-case execution time C_i. Since both values bound A_i, we take the minimum of them. ■

This estimation may be pessimistic, e.g., when $T_1 \gg C_1 > T_2 > 0$, as shown in Fig. 3 for the task set $\mathbb{T} = \{\tau_1, \tau_2\}$ with $\tau_1 = (C_1 = 4, S_1 = 0, D_1 = T_1 = 18)$ and $\tau_2 = (C_2 = 1, S_2 = 0, D_2 = T_2 = 3)$. If we estimate A_1 by $\widetilde{A}_1^1 = 3$, then we can bound $\widetilde{r}_2$ only by 4 which exceeds

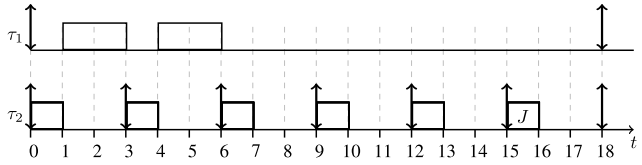

Fig. 3. For these tasks, Lemma 3 gives a bad estimation for A_1.

the relative deadline $T_2 = 3$. Actually, $A_1 = 0$ and $\widetilde{r}_2 = 1$ is smaller than T_2. We note that $\widetilde{r}_2 = 1$ is even a precise bound on the response time of τ_2.

If (an upper bound of) the response time R_i of J^i is known, A_i can be estimated with the following approach, that performs much better for the task set in Fig. 3:

Lemma 4: We can estimate A_i by

$$A_i \leq \max\left\{T_k + R_i - \left(\left\lfloor \frac{T_k}{T_i} \right\rfloor + 1\right) \cdot T_i, 0\right\} =: \widetilde{A}_i^2 \tag{9}$$

where R_i is either the worst-case response time of τ_i or any other upper bound on the response time of J^i.

Proof: Due to its definition, we know that the job J^i is released not later than $T_k - (\lfloor (T_k/T_i) \rfloor + 1) \cdot T_i$ and therefore must be finished not later than $T_k - (\lfloor (T_k/T_i) \rfloor + 1) \cdot T_i + R_i$, which may be less than 0 if J^i is actually finished before time 0. As a result, the maximum amount of time that J^i may interfere with the interval $[0, t] \subseteq [0, T_k]$ is bounded by the maximum of 0 or $T_k + R_i - (\lfloor (T_k/T_i) \rfloor + 1) \cdot T_i$, since the interference cannot be negative. ∎

The examination in this section leads to the following response time bound $\widetilde{R}_k$ which is not as tight as $\widetilde{r}_k$ but directly computable since it uses an estimation $\widetilde{A}_i$ of A_i.

Lemma 5: Let J be a job of τ_k and let all jobs with higher priority than J meet their deadline. Further, we define

$$\widetilde{R}_k := C_k + S_k + \sum_{i \neq k} \left(\left\lfloor \frac{T_k}{T_i} \right\rfloor \cdot C_i + \widetilde{A}_i \right) \tag{10}$$

where each $\widetilde{A}_i$ is defined as the minimum of $\widetilde{A}_i^1$ and $\widetilde{A}_i^2$ from (8) and (9), i.e.,

$$\widetilde{A}_i := \min\left\{C_i, \max\left\{T_k + R_i - \left(\left\lfloor \frac{T_k}{T_i} \right\rfloor + 1\right) \cdot T_i, 0\right\}\right\} \tag{11}$$

where R_i is an upper bound on the worst-case response time of τ_i or $R_i := T_i$. If $\widetilde{R}_k \leq T_k$, then J meets its deadline and $\widetilde{R}_k$ is an upper bound on the response time of J.

Proof: Lemmas 3 and 4 yield $\widetilde{r}_k \leq \widetilde{R}_k$. If $\widetilde{R}_k \leq T_k$, then we also have $\widetilde{r}_k \leq \widetilde{R}_k \leq T_k$ and by Proposition 2 the job J meets its deadline. The response time of J is upper bounded by $\widetilde{r}_k$ and hence, also by $\widetilde{R}_k$. ∎

Using this lemma, we obtain a response time bound valid for all jobs J of τ_k. Hence, $\widetilde{R}_k$ serves as a bound on the worst-case response time of τ_k and can be used for the estimation of $\widetilde{A}_k$ for other tasks in Lemma 5.

Theorem 4: Let $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$ be a sporadic task set whose tasks have implicit deadline. If for all $\tau_k \in \mathbb{T}$ the value $\widetilde{R}_k$ from (10) is at most T_k, then the task set is schedulable by preemptive EDF.

Proof: We know that $\widetilde{R}_k \geq \widetilde{r}_k$ for all $k \in \{1, \ldots, n\}$. If $T_k \geq \widetilde{R}_k$ for all k, then also $T_k \geq \widetilde{r}_k$. Hence, by Proposition 2, the task set $\mathbb{T}$ is schedulable by preemptive EDF and the worst-case response time of each τ_k is upper bounded by $\widetilde{r}_k \leq \widetilde{R}_k$. ∎

We can further improve the test by the following observations. If we do not estimate A_i by C_i, then we assume that J^i interferes for the whole time interval until the deadline (for $\widetilde{A}_i^1$), or until the finishing time (for $\widetilde{A}_i^2$). Although those intervals intersect for different jobs, we add the interference several times, which is overly pessimistic.

Proposition 3: Let $I \subseteq \{1, \ldots, n\} - \{k\}$ and let R_i be a bound on the response time of J^i for all $i \in I$. If no such bound is given, set $R_i = T_i$. We can estimate $\sum_{i \in I} A_i$ as

$$\sum_{i \in I} A_i \leq \max\left\{\max_{i \in I}\left\{T_k + R_i - \left(\left\lfloor \frac{T_k}{T_i} \right\rfloor + 1\right)T_i\right\}, 0\right\}. \tag{12}$$

Proof: We call f_{J^i} the finishing time and d_{J^i} the absolute deadline of the job J^i from Definition 3. Moreover, we still assume that $r_J = 0$. The job $J^i, i \in I$ may interfere after the release of J only during $[0, \max\{\max_{i \in I}\{f_{J^i}\}, 0\}]$. Furthermore, the processor can only be occupied by one job at a time. Hence, the sum of all interference of $J^i, i \in I$ is bounded by $\sum_{i \in I} A_i \leq \max\{\max_{i \in I}\{f_{J^i}\}, 0\}$.

We know that $d_{J^i} \leq T_k - (\lfloor (T_k/T_i) \rfloor) T_i$ is an estimation of the absolute deadline of J^i and $f_{J^i} \leq T_k + R_i - (\lfloor (T_k/T_i) \rfloor + 1) T_i$ is an upper bound on the finishing time of J^i if an estimation R_i on the response time of J^i is given. Since J^i finishes on time, we have $f_{J^i} \leq d_{J^i}$ and the estimation for the absolute deadline is also valid for the finishing time. Using both estimations in $\sum_{i \in I} A_i \leq \max\{\max_{i \in I}\{f_{J^i}\}, 0\}$ concludes the proof. ∎

For a specific job J^i, the interval starting from 0 up to the threshold given by the right hand side of (12) could also contain the interference by other jobs which are not overlapping. We use this observation to improve the test.

Proposition 4: Let $i \in \{1, \ldots, n\} - \{k\}$. The interference by the jobs of τ_i after a threshold $m \in [0, T_k]$ is bounded by

$$\left\lceil \frac{T_k - m}{T_i} \right\rceil \cdot C_i. \tag{13}$$

Proof: The interference can only occur for tasks with deadline after m. This is only possible for the last $\lceil [(T_k - m)/T_i] \rceil$ jobs. They contribute an interference of at most C_i each. ∎

Both ideas can be used to define a better estimation $\widetilde{R}_k$ than in (10). With this new definition of $\widetilde{R}_k$, Lemma 5 and Theorem 4 still hold. In the following we formulate this improved schedulability test.

A. Response-Time-Based Schedulability Test

Let $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$ be a sporadic implicit-deadline task set. Algorithm 1 is a sufficient test for the schedulability of $\mathbb{T}$ by preemptive EDF scheduling. If Algorithm 1 returns *schedulable*, then all $\widetilde{R}_k \leq T_k$, i.e., the task set $\mathbb{T}$ is schedulable according to Theorem 4 and its improvements in Propositions 3 and 4.

The algorithm estimates the response time of task τ_k for $k = n, \ldots, 1$. At first, in lines 4–6, it calculates $\widetilde{A}_i^k := T_k + R_i - (\lfloor (T_k/T_i) \rfloor + 1) T_i$ from Lemma 4 where R_i is an already computed response time bound (for $i > k$) or

Algorithm 1 Response-Time-Based Schedulability Test

1: Sort $\tau_1, \ldots, \tau_n$, such that $T_1 \leq \cdots \leq T_n$.
2: **for** $k = n, \ldots, 1$ **do**
3: $I^k := \{1, \ldots, n\} - \{k\}$
4: **for** $i \in I^k$ **do** ▷ Carry-in estimation from Lemma 4.
5: $$\widetilde{A}_i^k := \begin{cases} T_k - \left\lfloor \frac{T_k}{T_i} \right\rfloor \cdot T_i, & i < k \\ T_k + \widetilde{R}_i - \left(\left\lfloor \frac{T_k}{T_i} \right\rfloor + 1 \right) \cdot T_i, & i > k \end{cases}$$
6: **end for**
7: **for** $j \in I^k$ **do** ▷ Response time bounds for different thresholds.
8: $m_j^k := \max\left\{\widetilde{A}_j^k, 0\right\}$
9: $I_j^k := \left\{ i \in I^k \;\middle|\; \widetilde{A}_i^k \leq \widetilde{A}_j^k \right\}$
10: $$\widetilde{R}_k(j) := \sum_{i \in I^k - I_j^k} \min\left\{ \left\lfloor \frac{T_k}{T_i} \right\rfloor + 1, \left\lceil \frac{T_k - m_j^k}{T_i} \right\rceil \right\} C_i + \sum_{i \in I_j^k} \min\left\{ \left\lfloor \frac{T_k}{T_i} \right\rfloor, \left\lceil \frac{T_k - m_j^k}{T_i} \right\rceil \right\} C_i + C_k + S_k + m_j^k$$
11: **end for** ▷ Response time bound without threshold.
12: $\widetilde{R}_k(0) := C_k + S_k + \sum_{i \in I^k} \left(\left\lfloor \frac{T_k}{T_i} \right\rfloor + 1 \right) C_i$
13: $\widetilde{R}_k := \min\left\{ \widetilde{R}_k(j) \;\middle|\; j \in I^k \cup \{0\} \right\}$
14: **if** $\widetilde{R}_k > T_k$ **then**
15: **return** not schedulable
16: **end if**
17: **end for**
18: **return** schedulable

T_i. Then the algorithm tries out different thresholds m_j^k. For those tasks where A_i can be neglected (I_j^k) as in Proposition 3, it just adds the interference of the other $\lfloor (T_k/T_i) \rfloor$ jobs after the threshold, i.e., $\min\{\lfloor (T_k/T_i) \rfloor, \left\lceil [(T_k - m_j^k)/T_i] \right\rceil\} C_i$ due to Proposition 4. For the other tasks ($I^k - I_j^k$) we estimate A_i by C_i with the bound from Lemma 3. Together with Proposition 4 we obtain $\min\{\lfloor (T_k/T_i) \rfloor + 1, \left\lceil [(T_k - m_j^k)/T_i] \right\rceil\} C_i$. The algorithm then calculates the response time bound $\widetilde{R}_k(j)$ by adding the threshold m_j^k, the interference after the threshold, and $C_k + S_k$. Finally, the algorithm computes a response time bound $\widetilde{R}_k(0)$ for the threshold $m = 0$ (line 12). It derives the best bound by taking the minimum (line 13).

B. Theoretical Evaluation

We compare the RTA presented in this section with the existing tests summarized in Section III.

Comparison With Suspension-Oblivious: Although we show in Section VI that our method has a better performance, both schedulability tests do not dominate each other. There are task sets which pass our schedulability test, although the suspension-oblivious approach, see Theorem 1, cannot give any schedulability guarantees.

Example 1: We assume that $\mathbb{T} = \{\tau_1, \tau_2\}$ consists of:

1) $\tau_1 = (C_1 = 1, S_1 = 2, D_1 = T_1 = 5)$;
2) $\tau_2 = (C_2 = 1, S_2 = 3, D_2 = T_2 = 7)$.

The worst-case response times for the tasks are bounded by:

1) $\widetilde{R}_2 = 1 + 3 + (\lfloor (7/5) \rfloor + 1) \cdot 1 = 6 \leq 7 = T_2$;
2) $\widetilde{R}_1 = 1 + 2 + (\lfloor (5/7) \rfloor + 1) \cdot 1 = 4 \leq 5 = T_1$.

Hence, $\mathbb{T}$ is schedulable according to our test. However, for the condition in Theorem 1 we obtain $[(1+2)/5] + [(1+3)/7] > 1$.

On the other hand, there are some task sets that are schedulable by Theorem 1 but not according to our test:

Example 2: The task set $\mathbb{T} = \{\tau_1, \tau_2\}$ with two tasks $\tau_1 = (C_1 = 3, S_1 = 0, D_1 = T_1 = 6)$ and $\tau_2 = (C_2 = 10, S_2 = 0, D_2 = T_2 = 20)$ is schedulable by Theorem 1. Our method however computes the value $\widetilde{R}_2 = 10 + \lfloor [(20-2)/6] \rfloor \cdot 3 + 2 = 21$ which is higher than the relative deadline of task τ_2.

Comparison With the Workload-Based Schedulability Test by Liu and Anderson: Comparing their estimations with the ones described in this section yields the following.

Proposition 5: If all tasks are self-suspending tasks, our method dominates the analysis by Liu and Anderson from [15] applied to implicit-deadline tasks in uniprocessor systems.

Proof: For contradiction, we assume that our schedulability test fails but the one described in Theorem 2 succeeds. At first, we want to find a possible value for ξ_l. If the algorithm described in Section IV-A stops without determining the schedulability of the task set, then there is some $l \in \{0, 1, \ldots, n\}$ with $\widetilde{R}_l > T_l$. We deduce that $T_l < \widetilde{R}_l \leq \widetilde{R}_l(0) = C_l + S_l + \sum_{i \neq l} (\lfloor (T_l/T_i) \rfloor + 1) \cdot C_i \leq C_l + S_l + T_l \sum_i (C_i/T_i) + \sum_i C_i$. This implies $C_l + S_l + \sum_i C_i > T_l \cdot (1 - \sum_i (C_i/T_i))$ and hence $[(C_l + S_l + \sum_i C_i)/(1 - \sum_i (C_i/T_i))] > T_l$. We conclude that the interval $[T_l, [(C_l + S_l + \sum_i C_i)/(1 - \sum_i (C_i/T_i))])$ is not empty and $\xi_l = T_l$ has to be considered in the schedulability test according to Theorem 2.

By setting $\xi_l = T_l$ and $s_{l,j} = S_l$, we obtain

$$\sum_{\tau_i \in \tau^s} \max\{W_{nc}(\tau_i), W_c(\tau_i)\} + \sum_{\tau_j \in \tau^e} W_{nc}(\tau_j) \leq T_l - C_l - S_l. \quad (14)$$

Furthermore, we know that the set τ^e is empty and $\tau^s = \mathbb{T}$ by assumption. Hence, the LHS of inequality (14) is $\sum_{\tau_i \in \mathbb{T}} \max\{W_{nc}(\tau_i), W_c(\tau_i)\}$ and we can estimate it from below by $W_{nc}(\tau_l) + \sum_{\tau_i \neq \tau_l} W_c(\tau_i) \leq LHS$.

The computation of $W_{nc}(\tau_l)$ yields that $W_{nc}(\tau_l) = \min\{\lfloor (T_l/T_l) \rfloor C_l - C_l, T_l - T_l\} = 0$, and $W_c(\tau_i), i \neq l$ is defined by $W_c(\tau_i) = \min\{\Delta(\tau_i, T_l), T_l - C_l - S_l + 1\}$ with $\Delta(\tau_i, T_l) = (\lceil (T_l/T_i) \rceil - 1) C_i + \min\{C_i, T_l - \lceil (T_l/T_i) \rceil T_i + T_i\}$.

By using the four inequalities[5] $T_l - C_l - S_l + 1 \geq 0$, $(\lceil (T_l/T_i) \rceil - 1) \geq 0$, $C_i \geq 0$ and $T_l - \lceil (T_l/T_i) \rceil T_i + T_i \geq T_l - ((T_l/T_i) + 1) T_i + T_i = 0$, we obtain that $W_c(\tau_i)$ is not negative for all $i \neq l$. We conclude that $W_c(\tau_i) = \Delta(\tau_i, T_i)$ since otherwise the LHS from (14) would be at least $T_l - C_l - S_l + 1$ and hence $T_l - C_l - S_l + 1 \leq LHS \leq T_l - C_l - S_l$, which is a contradiction.

In the following, we show that even:

$$\Delta(\tau_i, T_l) = \left\lfloor \frac{T_l}{T_i} \right\rfloor C_i + \min\left\{ C_i, T_l - \left\lfloor \frac{T_l}{T_i} \right\rfloor T_i \right\} \quad (15)$$

holds. For all T_l which are not in $\mathbb{Z}T_i$, we know that $\lceil (T_l/T_i) \rceil - 1 = \lfloor (T_l/T_i) \rfloor$. By using this in the definition of $\Delta(\tau_i, T_l)$, we obtain the result from (15). In the other case $T_l \in \mathbb{Z}T_i$, we have $\lceil (T_l/T_i) \rceil = (T_l/T_i) = \lfloor (T_l/T_i) \rfloor$ which yields $\Delta(\tau_i, T_l) = ((T_l/T_i) - 1) C_i + \min\{C_i, T_l - (T_l/T_i) \cdot T_i + T_i\} = (T_l/T_i) C_i = \lfloor (T_l/T_i) \rfloor C_i + \min\{C_i, T_l - \lfloor (T_l/T_i) \rfloor T_i\}$.

[5] We know that $T_l \geq C_l + S_l$ since otherwise $\mathbb{T}$ is not schedulable and both schedulability tests would fail.

We conclude that if the test by Liu and Anderson as in Theorem 2 indicates schedulability of the task system, then the inequality $C_l+S_l+\sum_{\tau_i\neq\tau_l}\lfloor(T_l/T_i)\rfloor C_i+\min\{C_i, L(i)\} \leq T_l$ with $L(i) = T_l - \lfloor(T_l/T_i)\rfloor T_i$ holds. It is left to show that from this inequality also $\widetilde{R}_l \leq T_l$ follows.

Let $I \coloneqq \{i \in \{1, 2, \ldots, n\} \mid i \neq l, L(i) \leq C_i\}$. If the set I is empty, then $\widetilde{R}_l(0) = C_l+S_l+\sum_{\tau_i\neq\tau_l}\lfloor(T_l/T_i)\rfloor C_i+\min\{C_i, L(i)\}$ and we get $\widetilde{R}_l \leq \widetilde{R}_l(0) \leq T_l$, which contradicts the assumption that $\widetilde{R}_l > \widetilde{T}_l$. Otherwise, if I is not empty, we define $\tilde{j} \coloneqq \arg\max_{i\in I}\{A_i^l\}$. If there is not a unique integer for the maximum, we choose one among them. In the following we show that the right hand side of $\widetilde{R}_l(\tilde{j}) \leq C_l + S_l + \sum_{\tau_i\neq\tau_l}\lfloor(T_l/T_i)\rfloor C_i + \sum_{i\in I^l - I_{\tilde{j}}^l} C_i + \max\{A_{\tilde{j}}^l, 0\}$ is at most $C_l + S_l + \sum_{\tau_i\neq\tau_l}(\lfloor(T_l/T_i)\rfloor C_i + \min\{C_i, L(i)\})$. The term $C_l + S_l + \sum_{\tau_i\neq\tau_l}\lfloor(T_l/T_i)\rfloor C_i$ exists in both expressions. Therefore, we consider only the remaining part $\sum_{i\in I^l - I_{\tilde{j}}^l} C_i + \max\{A_{\tilde{j}}^l, 0\}$.

Since $A_{\tilde{j}}^l \leq L(\tilde{j}) \leq C_{\tilde{j}}$ and $0 \leq L(\tilde{j})$, we have $\max\{A_{\tilde{j}}^l, 0\} \leq L(\tilde{j}) = \min\{C_{\tilde{j}}, L(\tilde{j})\}$ for the second summand. Furthermore, if $i \in I^l - I_{\tilde{j}}^l$, then by definition $A_i^l > A_{\tilde{j}}^l$ and $i \notin I$. Hence, we obtain $L(i) > C_i$ and $\sum_{i\in I^l - I_{\tilde{j}}^l} C_i = \sum_{i\in I^l - I_{\tilde{j}}^l} \min\{C_i, L(i)\} \leq \sum_{i\neq l, i\neq\tilde{j}} \min\{C_i, L(i)\}$ for the first summand. We conclude that in the case $I \neq \emptyset$, the inequality $\widetilde{R}_l(\tilde{j}) \leq C_l + S_l + \sum_{i\neq l}\lfloor(T_l/T_i)\rfloor C_i + \sum_{i\neq l} \min\{C_i, L(i)\} \leq T_l$ also holds, which contradicts our assumption. ∎

We note that we did not examine, whether our schedulability test still dominates the method by Liu and Anderson [15] if some tasks are execution-only (i.e., $\tau^e \neq \emptyset$).

C. Remarks on Extensions

For a more rigorous proof and dominance discussion we focused our attention on the fundamental case with implicit-deadline uniprocessor systems. Nevertheless, with minor adjustment, the RTA can also be applied to constrained-deadline sporadic real-time task sets, in which $D_i \leq T_i$ for every τ_i. Furthermore, this analysis which is based on (7) in Proposition 2 can be modified to handle global EDF on M processors as follows:

$$\widetilde{r}_k \coloneqq S_k + C_k + \frac{\sum_{i\neq k}\left(\left\lfloor \frac{T_k}{T_i} \right\rfloor \cdot C_i + A_i\right)}{M} \leq T_k. \qquad (16)$$

That is, 1) when task τ_k suspends, no job is executed on any of the M processors and 2) when task τ_k executes on one processor all $M-1$ processors idle.

V. Method 2: Redundant Self-Suspension Analysis

Our second approach is an improvement of the suspension-oblivious schedulability test, which is stated in Theorem 1. The underlying utilization-based test [16] is exact for periodic tasks without suspension, i.e., the test succeeds if and only if the task set is schedulable. The overall aim of this section is to provide a suspension-aware version of that test.

The central observation is that the processor can be suspended for several jobs and additionally work on a lower priority job at the same time, as already mentioned in Remark 1. The suspension-oblivious approach is not aware of this fact and adds the time where the processor is suspended for a job as additional execution time to the workload. In the following, we estimate the number of jobs by which the processor is only suspended while it is also suspended for or working on lower priority jobs, and exclude their suspension time from the workload. To make the estimations possible we have to restrict to periodic task sets, where time between recurrent job releases is exactly the period. We note that by omitting the implicit-deadline restriction our analysis would become incorrect, too.

Let $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$ be a periodic implicit-deadline task system. We consider an EDF-schedule of jobs released by this task system. Furthermore, we assume that it is not schedulable, i.e., that there exists a job in the schedule that does not meet its deadline.

A. Preparation

Since we consider a periodic task system, the release pattern of the jobs is fixed, depending on the given phases. For a given schedule, we reorganize the task system and the schedule to fulfill the following conditions.

Remark 3 (Revision of the EDF Schedule):

1) The property $C_1 + S_1 \leq \cdots \leq C_n + S_n$ holds. This is accomplished simply by reindexing.
2) For each task, no additional job at the beginning can be added without preserving the minimal phase, i.e., for $\phi \coloneqq \min\{\phi_i\}$ the property $\phi_i - \phi < T_i$ holds for all i. More specifically, $\phi_i \in [\phi, \phi + T_i]$ for all i. This is achieved by adding empty jobs without any execution or suspension time of τ_i periodically between ϕ_i and $\min\{\phi_i\}$ until $\phi_i - \phi < T_i$ holds.
3) All jobs have *full* execution and suspension time, i.e., the processor is working on and suspended for each job the amount of time given by the worst-case execution and maximum suspension time. This is achieved by the undermentioned procedure. Lemma 6 proves that this step does not transform any EDF schedule with deadline misses into an EDF schedule without deadline misses.

We note that the addition of empty jobs in step (2) has no impact on the response times of the other jobs. After ensuring Property (2) we denote the jobs in the schedule by $J_1, J_2, \ldots$, sorted by their priorities. Inductively, we modify the schedule again by using the following procedure for the mth job J_m, $m = 1, 2, \ldots$, to obtain Property (3).

1) If the processor is now working on any of the jobs $J_1, \ldots, J_{m-1}$ at a time frame where it was before suspended for or working on J_m, we move this execution or suspension time of J_m to the end of that job.
2) We add additional (pseudo) execution and suspension time at the end of J_m such that it equals the worst case.

Lemma 6: By using the procedure described above, the response time of each job does not decrease.

Proof: We use induction and prove that after the mth step, $J_1, \ldots, J_m$ interfere at least the same time frames as before and the response time of J_m is not decreased.

For the first job, we can just add additional suspension and execution time at the end. This can not decrease the response

time of that job and J_1 interferes at least the same time frames as before, possibly even more.

A modification of the mth job does not affect the response times of the higher priority jobs $J_1, \ldots, J_{m-1}$. By induction, those higher priority jobs interfere for at least the same time frames as before. Therefore, and because we only add execution and suspension time, the response time of J_m does not decrease. If the processor was before suspended for or working on J_m, then it is now either still in the same state or it is occupied by any of the jobs $J_1, \ldots, J_{m-1}$. Hence, the processor is occupied by $J_1, \ldots, J_m$ at least at the same time frames as before, i.e., the interference by $J_1, \ldots, J_m$ may only be increased. ■

By Lemma 6, if the original EDF schedule has a deadline miss, the new EDF schedule with its jobs in the setting described in Remark 3 also has a deadline miss.

B. Proof of the Schedulability Test

We assume that the requirements from Remark 3 are met by the task set. Let J be the job with the highest priority which does not meet its deadline. Let this deadline be t_d. Furthermore, let t_{-1} be the last time before t_d where the processor is not suspended, or working on J or jobs with higher priority. We delete all jobs which are finished before t_{-1} or have lower priority than J. This does not affect the response times of the remaining jobs.

For notational brevity, the remaining schedule from time t_{-1} to t_d is called σ for the rest of the proof. According to the above definition, the jobs that are executed in the time interval $[t_{-1}, t_d]$ have release time $\geq t_{-1}$ and deadline $\leq t_d$. Hence, there are at most $\lfloor [(t_d - t_{-1})/T_i] \rfloor$ jobs of task τ_i in the schedule σ.

Moreover, by the above definition of the interval $[t_{-1}, t_d]$ and the schedule σ, the processor is either working on a job or suspended at any time point from t_{-1} to t_d. Suppose that V_{run} (V_{sus}, respectively) is the amount of time that the processor is working (suspended, respectively) from t_{-1} to t_d. Since one job misses its deadline, we know that

$$t_d - t_{-1} = V_{\text{run}} + V_{sus} < \sum_{\tau_i \in \mathbb{T}} \left\lfloor \frac{t_d - t_{-1}}{T_i} \right\rfloor C_i + \sum_{\tau_i \in \mathbb{T}} \left\lfloor \frac{t_d - t_{-1}}{T_i} \right\rfloor S_i. \quad (17)$$

The condition $t_d - t_{-1} < \sum_{\tau_i \in \mathbb{T}} \lfloor [(t_d - t_{-1})/T_i] \rfloor (C_i + S_i)$ implies that $1 < \sum_{\tau_i \in \mathbb{T}} [(C_i + S_i)/T_i]$, which concludes the suspension-oblivious schedulability test in Theorem 1.

Instead of applying $V_{sus} \leq \sum_{\tau_i \in \mathbb{T}} \lfloor [(t_d - t_{-1})/T_i] \rfloor S_i$, our utilization-based analysis intends to construct a tighter bound for V_{sus} and is based on the following observation.

Observation 3: Suppose that there are two jobs J and $J^\dagger$ in schedule σ. If we have $[r_J, d_J] \subseteq [r_{J^\dagger}, r_{J^\dagger} + R_{J^\dagger}]$, where r_J (d_J) is the release time (deadline) of J and $R_{J^\dagger}$ is the response time of $J^\dagger$, then the suspension time of job J only contributes to the suspension in the schedule σ when $J^\dagger$ suspends at the same time. In this case, the suspension of the processor due to J is already covered by the suspension of $J^\dagger$ and does not have to be considered in the estimation of V_{sus}.

Let l be the highest index among those tasks which do have remaining jobs after these deletions. Let t^l_{-1} and t^l_d be the first release and last deadline of the remaining jobs of τ_l, respectively. If $l = 1$, then there is only one task τ_l in the remaining schedule in the time interval $[t_{-1}, t_d]$ and the deadline miss of τ_l implies that $C_l + S_l > D_l = T_l$, which violates our assumption that $C_l + S_l \leq D_l$. We therefore only have to consider the scenario where $l \geq 2$.

For each of the $[(t^l_d - t^l_{-1})/T_l]$ jobs, we utilize it as $J^\dagger$ in Observation 3 and then quantify the jobs J that satisfy the condition $[r_J, d_J] \subseteq [r_{J^\dagger}, r_{J^\dagger} + R_{J^\dagger}]$.

Lemma 7: Suppose that job $J^\dagger$ is a job of τ_l in the schedule σ and $C_l + S_l \geq T_i$ for a certain task τ_i. Then, at least $\lfloor [(C_l + S_l)/T_i] \rfloor - 1$ different jobs of task τ_i are jobs J that satisfy the condition in Observation 3.

Proof: Since the execution time plus the suspension time of job $J^\dagger$ is $C_l + S_l$ in the schedule σ due to Property (3) in Remark 3, the response time of job $J^\dagger$ is at least $C_l + S_l$. Moreover, since task τ_i is periodically released with Property (2) in Remark 3, the first job release of task τ_i after $r_{J^\dagger}$ must be earlier than $r_{J^\dagger} + T_i$. Therefore, there are at least $\left\lfloor [(r_{J^\dagger} + C_l + S_l - (r_{J^\dagger} + T_i))/T_i] \right\rfloor = \lfloor [(C_l + S_l)/T_i] \rfloor - 1$ jobs of task τ_i with release time $\geq r_{J^\dagger}$ and deadline $\leq r_{J^\dagger} + C_l + S_l$. ■

We already calculated the number of jobs of τ_l in the schedule σ by $[(t^l_d - t^l_{-1})/T_l]$. However, to perform a similar proof as for the suspension-oblivious test in (17), the term $(t^l_d - t^l_{-1})$ is impractical and we need to cancel out $(t_d - t_{-1})$ instead. Hence, we estimate $(t^l_d - t^l_{-1})$ with respect to $(t_d - t_{-1})$.

Lemma 8: The ratio of $t^l_d - t^l_{-1}$ to $t_d - t_{-1}$ is at least $(1/3)$.

Proof: Since the other jobs are deleted, we have $[t^l_{-1}, t^l_d] \subseteq [t_{-1}, t_d]$. If the remaining part on the right side of the interval would be more than T_l, i.e., $t_d - t^l_d > T_l$, then there would be another job of τ_l released at t^l_d with higher priority than J, which does not exist by the definition of t^l_d. Furthermore, if $t^l_{-1} - t_{-1} > T_l$, then there would be enough space for another job with deadline at t^l_{-1} and release time after t_{-1}. By Property (2) in Remark 3, the phase of τ_l is smaller than t^l_{-1} and the additional job in fact exists. This contradicts the definition of t^l_{-1}. We conclude that $[t^l_{-1}, t^l_d] \subseteq [t_{-1}, t_d] \subseteq [t^l_{-1} - T_l, t^l_d + T_l]$.

Let h be the number of jobs of τ_l in σ defined by the quotient $[(t^l_d - t^l_{-1})/T_l]$, then we obtain $[(t^l_d - t^l_{-1})/(t_d - t_{-1})] \geq [(h \cdot T_l)/(h \cdot T_l + 2 \cdot T_l)] = [h/(h+2)]$. Since $h \in \mathbb{Z}_{\geq 1}$ and $[h/(h+2)] \leq [((h+1))/((h+1)+2)]$ for all $h \in \mathbb{Z}_{\geq 1}$, we get $[h/(h+2)] \geq (1/3)$ and $[(t^l_d - t^l_{-1})/(t_d - t_{-1})] \geq (1/3)$. ■

Now, we can conclude the schedulability test.

Theorem 5: Let $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$ be a set of implicit-deadline periodic tasks with dynamic self-suspension and $C_1 + S_1 \leq \cdots \leq C_n + S_n$. For notational brevity, let δ^l_i denote the indication function which is 1 if $C_l + S_l \geq T_i$ and 0 otherwise. If for all $l \in \{1, \ldots, n\}$ the property

$$\frac{C_l + S_l}{T_l} + \sum_{i=1}^{l-1} \frac{C_i + S_i \left(1 - \frac{1}{3}\frac{T_i}{T_l}\left(\left\lfloor \frac{C_l + S_l}{T_i} \right\rfloor - 1\right) \cdot \delta^l_i\right)}{T_i} \leq 1 \quad (18)$$

holds, then the task set is schedulable by preemptive EDF. We set $\sum_{i=1}^{l-1} X_i = 0$ when l is 1 for notational brevity.

Proof: We prove this theorem by contraposition. Suppose that there is a job missing its deadline, and we construct the remaining schedule σ in time interval $[t_{-1}, t_d]$ as defined above, in which a job misses its deadline in the schedule σ. Let l be the highest index among those tasks with at least one job in schedule σ. We define $C'_i \coloneqq C_i + S_i$ for all i.

We first focus on the scenario when $l \geq 2$. Let h be the number of jobs of task τ_l in the time interval $[t_{-1}, t_d]$, i.e., h is $[(t_d^l - t_{-1}^l)/T_l]$. By Lemma 8 and the definition of periodic tasks, we have $[(t_d - t_{-1})/3T_l] \leq h \leq [(t_d - t_{-1})/T_l]$.

We can use Lemma 7 and Observation 3 to reduce the contribution of the suspension time to V_{sus} from a certain task τ_i for $i = 1, 2, \ldots, l-1$. Together with an argument similar to that resulting in the condition in (17), we have $t_d - t_{-1} < hC'_l + \sum_{i=1}^{l-1}(\lfloor[(t_d - t_{-1})/T_i]\rfloor C'_i - h(\lfloor (C'_l/T_i)\rfloor - 1)\delta_i^l S_i) \leq [(t_d - t_{-1})/T_l]C'_l + \sum_{i=1}^{l-1}([(t_d - t_{-1})/T_i]C'_i - [(t_d - t_{-1})/3T_l](\lfloor (C'_l/T_i)\rfloor - 1)\delta_i^l S_i) =: A$. The second inequality is due to $[(t_d - t_{-1})/3T_l] \leq h \leq [(t_d - t_{-1})/T_l]$.

Moreover, when l is 1, the schedule σ has only jobs from τ_1 in $[t_{-1}, t_d]$. That means $t_d - t_{-1} < h \cdot C'_l \leq [(t_d - t_{-1})/T_l] \cdot C'_l$, i.e., the condition $t_d - t_{-1} \leq A$ holds also when $l = 1$.

Therefore, by dividing both sides of $t_d - t_{-1} \leq A$ with $t_d - t_{-1}$, which is > 0, the deadline miss of a job in the schedule σ implies the existence of $l = 1, 2, \ldots, n$ and $1 < (C'_l/T_l) + \sum_{i=1}^{l-1}(C_i/T_i) + [(S_i(1 - (1/3)(T_i/T_l)(\lfloor (C'_l/T_i)\rfloor - 1) \cdot \delta_i^l))/T_i]$. As a result, the negation of the above condition, i.e., for all $l = 1, 2, \ldots, n$ with the condition in (18), implies the schedulability of the task set under preemptive EDF. ∎

C. Theoretical Evaluation

We provide some observations about the theoretical behavior of the schedulability test provided in this section in comparison to the prior methods detailed in Section III and our RTA in Section IV.

Comparison With Suspension-Oblivious: The redundant self-suspension schedulability test proposed in Theorem 5 dominates the existing utilization-based suspension-oblivious analysis applied to periodic task sets, since the LHS of (18) is at most the total utilization if we consider suspension time as additional execution time. Our method performs better if one task has a big sum of suspension and execution time compared to the period of the other tasks.

Example 3: For the task set $\mathbb{T} = \{\tau_1, \tau_2\}$ with two tasks $\tau_1 = (C_1 = (1/17), S_1 = (1/3), D_1 = T_1 = 1)$ and $\tau_2 = (C_2 = 14, S_2 = 0, D_2 = T_2 = 21)$ suspension-oblivious analysis concludes that the task set is not schedulable, since the total utilization is > 1. Our test however indicates schedulability: At first we observe that for $l = 1$, we have $[(C_1 + S_1)/T_1] = [((1/17) + (1/3))/1] = (20/51) \leq 1$. For $l = 2$, we compute $(1/3)(T_1/T_2)(\lfloor[(C_2 + S_2)/T_1]\rfloor - 1) = (1/3)(1/21) \cdot 13 = (13/63)$. The test in (18) yields $(14/21) + [((1/17) + (1/3)(1 - (13/63)))/1] \leq 1$. Hence, the task set is schedulable.

Comparison With Response Time Analysis in Section IV: Both schedulability tests proposed in this article do not dominate each other. To prove this, we again use Example 1 and Example 2 with periodic task sets. Since neither $C_1 + S_1 \geq T_2$ nor $C_2 + S_2 \geq T_1$, our utilization-based approach coincides with the suspension-oblivious schedulability test, which is superior in Example 2 and inferior in Example 1.

Comparison With the Workload-Based Schedulability Test by Liu and Anderson: To show that the schedulability test by Liu and Anderson [15] and the one described in this section also do not dominate each other, we use similar examples with periodic task sets.

The method by Liu and Anderson is superior for Example 1. For $l = 1$ only $s_{l,j} = 2$ and $\xi_l = 5$ is possible. By computation we obtain $W_c(\tau_1) = 0, W_{nc}(\tau_1) = 0, W_c(\tau_2) = 1$, and $W_{nc}(\tau_2) = 0$. Since $1 \leq 5 - 1 - 2 = 2$, (1) holds. For $l = 2$ no choice of $s_{l,j}$ and ξ_l is possible. Hence, the test says that the task set is schedulable. Since $C_2 + S_2 = 4 < 5 = T_1$, our schedulability test is the same as the suspension-oblivious approach. Therefore, our test can not conclude schedulability.

For a task set that is similar to the one presented in Example 2, our method is superior to the one by Liu and Anderson: Let $\tau_1 = (C_1 = 3 - \varepsilon, S_1 = \varepsilon, D_1 = T_1 = 6)$ and $\tau_2 = (C_2 = 10 - \varepsilon, S_2 = \varepsilon, D_2 = T_2 = 20)$ with $\varepsilon \in (0, (1/3))$. The periodic task set is not deemed schedulable by our response-time-based analysis in Section IV. Since it dominates the schedulability test by Liu and Anderson [15] if all tasks have suspension as discussed in Section IV-B, their test also fails for this example. We have $\widetilde{R}_2(0) = 10 + (\lfloor(20/6)\rfloor + 1) \cdot (3 - \varepsilon)$ and $\widetilde{R}_2(1) = 10 + \lfloor[(20 - 2)/6]\rfloor \cdot (3 - \varepsilon) + 2$. Hence, the value for $\widetilde{R}_2$ is $10 + 3 \cdot (3 - \varepsilon) + 2 = 21 - 3\varepsilon$ which is bigger than 20 for $\varepsilon < (1/3)$. Our response-time-based analysis in Section IV as well as the approach by Liu and Anderson [15] both fail.

On the other hand we know that the example is schedulable by the suspension-oblivious approach since the value for $\sum_i [(C_i + S_i)/T_i]$ does not change compared to Example 2. Since our redundant self-suspension schedulability test dominates the suspension-oblivious test, it also implies schedulability.

VI. Experimental Evaluation

We evaluated synthesized task sets to compare the proposed methods with the approaches from the literature. The metric to compare is the *acceptance ratio* with respect to the task set utilization, i.e., the percentage of task sets that has been accepted for the specific utilization value. We generated 1000 task sets for each of the analyzed utilization levels in the range of 0% to 100% with steps of 1%.

For each task set, we first generated a set of implicit-deadline tasks. We adopted the UUniFast method [3] to generate a set of utilization values U_i with the given goal U_{sum}, and applied the suggestion by Emberson *et al.* [13] to generate the periods of the tasks according to a log-uniform distribution with two orders of magnitude. To be precise, $\log_{10} T_i$ is a uniform distribution and the resulting T_i values are in [1ms, 100ms]. Accordingly, the execution time was set to $C_i = T_i \cdot U_i$ and the relative deadline was set to the task periods, i.e., $D_i = T_i$. We converted them to dynamic self-suspending

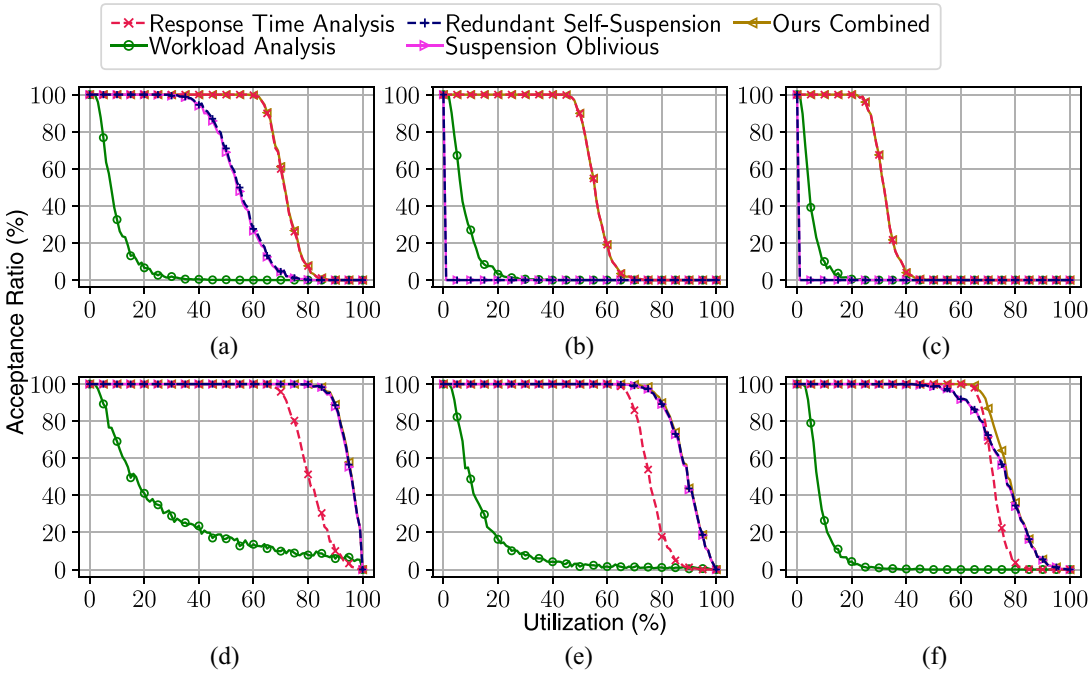

Fig. 4. Comparison of our schedulability tests with the state-of-the-art. (a) Uniform suspension in [0.0-0.1], 10 tasks. (b) Uniform suspension in [0.1-0.3], 10 tasks. (c) Uniform suspension in [0.3-0.6], 10 tasks. (d) Log-uniform suspension in [0.0001-0.1], 5 tasks. (e) Log-uniform suspension in [0.0001-0.1], 10 tasks. (f) Log-uniform suspension in [0.0001-0.1], 20 tasks.

tasks by generating the suspension lengths of each task according to a uniform distribution in either of three ranges, i.e., the targeted self-suspension length.

1) *Short Suspension:* $[0.0(T_i - C_i), 0.1(T_i - C_i)]$.
2) *Moderate Suspension:* $[0.1(T_i - C_i), 0.3(T_i - C_i)]$.
3) *Long Suspension:* $[0.3(T_i - C_i), 0.6(T_i - C_i)]$.

Furthermore, we considered log-uniformly distributed self-suspension in $[0.0001(T_i - C_i), 0.1(T_i - C_i)]$. We compared the following methods.

1) Our RTA detailed in Section IV.
2) Our utilization-based analysis that examines *redundant self-suspension* (Section V).
3) Combination of the above methods, denoted by *ours combined*, i.e., if one of them returns *schedulable*, this test succeeds.
4) The *suspension oblivious* approach, see Section III-A.
5) The *workload analysis* by Liu and Anderson [15], see Section III-B.
6) The *utilization-based method* by Dong and Liu [12], see Section III-C.

The results with uniformly distributed self-suspension with ten tasks per task set are shown in Fig. 4(a)–(c), and the results with log-uniformly distributed self-suspension are shown in Fig. 4(d)–(f) with 5, 10 and 20 tasks per task set. The *utilization-based method* is omitted, since it provided the same results as the *suspension oblivious* approach. While for comparison with the *redundant self-suspension* method, we assume that all tasks are periodic, comparison between the other methods is based even on the sporadic counterparts.

The RTA clearly outperforms the state-of-the-art for all cases with uniform distributed self-suspension length as depicted in Fig. 4(a)–(c). As expected, the acceptance ratio is reduced for all schedulability test if the suspension interval gets longer. The *workload analysis* by Liu and Anderson [15] always drops very quickly, but, contrary to the utilization-based analyses, still accepts some task sets for longer suspension intervals. The *redundant self-suspension* and the *suspension oblivious* approach only perform reasonably good for short suspension as in Fig. 4(a).

Redundant self-suspension and *suspension oblivious* show enhanced performance for short log-uniformly distributed self-suspension as in Fig. 4(d)–(f). Interestingly, we can see that those results depend on the number of tasks. The less tasks are in the task set, the better performs *redundant self-suspension* compared to RTA, i.e., in Fig. 4(d) and (e) the former provides better results while in Fig. 4(f) it depends on the utilization.

The result of *ours-combined* shows another aspect of the relation of our proposed methods. For Fig. 4(a)–(e) its curve lies almost directly on the individually better performing curve, i.e., the largest gap is 1.3%. On the other hand for Fig. 4(f) we observe additional benefit for combining both tests of up to 14.6% increase of acceptance ratio.

As proven in Section V, the *redundant self-suspension* method dominates the *suspension oblivious* approach. Table I summarizes the acceptance ratio gain, where the gain of acceptance ratio of A compared to B is computed by subtracting the acceptance ratio of B from the acceptance ratio of A. in different utilization ranges of experiments (d)–(f). We further examine how this improvement changes if the period range is extended from [1, 100] to [1, 10000], indicated by (d)'–(f)'. In almost all cases, the average acceptance ratio gain increases. We note that the extended periods do not significantly impact the performance of the schedulability test compared to each other which is why the plots for (d)'–(f)' are omitted. As

TABLE I
AVERAGE ACCEPTANCE RATIO GAIN [%] OF REDUNDANT SELF-SUSPENSION COMPARED TO SUSPENSION OBLIVIOUS. (d)–(f) ARE EXPERIMENTS FROM FIG. 4 AND (d)'–(f)' ARE THE SAME EXPERIMENTS WITH PERIODS PULLED FROM [1, 10000]

Utilization range [%]	(d)	(e)	(f)	(d)'	(e)'	(f)'
1-10	0.00	0.00	0.00	0.00	0.00	0.00
11-20	0.00	0.00	0.00	0.00	0.00	0.00
21-30	0.00	0.00	0.01	0.00	0.00	0.00
31-40	0.00	0.00	0.02	0.00	0.00	0.02
41-50	0.00	0.00	0.02	0.00	0.00	0.11
51-60	0.00	0.00	0.16	0.00	0.01	0.53
61-70	0.00	0.02	0.39	0.00	0.08	1.26
71-80	0.00	0.19	0.52	0.01	0.74	1.37
81-90	0.39	0.79	0.25	0.69	1.89	0.69
91-100	0.90	0.31	0.02	1.44	0.79	0.03

depicted in the table, an average acceptance ratio gain of up to 1.89% is observed.

VII. CONCLUSION

Despite the importance of preemptive dynamic-priority scheduling, there is no schedulability test dedicated to suspension-aware analysis for the dynamic self-suspension model in uniprocessor systems. The only applicable methods known from the literature are the methods by Liu and Anderson [15] and Dong and Liu [12] that were originally designed for global EDF and are also valid for uniprocessor systems. We propose two schedulability tests that they improve those state-of-the-art methods for uniprocessor systems, both theoretically and in the evaluation. In the future, we plan to further explore the necessary schedulability conditions and tighten the sufficient schedulability conditions. We will also explore the computational complexity of deriving an exact schedulability test for implicit-deadline periodic/sporadic task systems with the dynamic suspension under preemptive EDF, which has been noted as an open problem in Section VIII-B2 in the review paper by Chen *et al.* [9].

ACKNOWLEDGMENT

The authors thank Prof. Cong Liu for acknowledging a typing error in the formulation of the schedulability test for EDF in [15] and for confirming the correct formulation used in Theorem 2.

REFERENCES

[1] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proc. 11th Real-Time Syst. Symp. (RTSS)*, 1990, pp. 182–190.

[2] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Proc. Real Time Syst. Symp. (RTSS)*, 2007, pp. 149–160.

[3] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real Time Syst.*, vol. 30, nos. 1–2, pp. 129–154, 2005.

[4] B. B. Brandenburg. (2019). *Multiprocessor Real-Time Locking Protocols: A Systematic Review*. [Online]. Available: http://arxiv.org/abs/1909.09600

[5] J.-J. Chen, "Computational complexity and speedup factors analyses for self-suspending tasks," in *Proc. Real Time Syst. Symp. (RTSS)*, 2016, pp. 327–338.

[6] J.-J. Chen, T. Hahn, R. Hoeksma, N. Megow, and G. von der Brüggen, "Scheduling self-suspending tasks: New and old results," in *Proc. Euromicro Conf. Real Time Syst.*, 2019, pp. 1–23.

[7] J.-J. Chen and C. Liu, "Fixed-relative-deadline scheduling of hard real-time tasks with self-suspensions," in *Proc. Real Time Syst. Symp. (RTSS)*, 2014, pp. 149–160.

[8] J.-J. Chen, G. Nelissen, and W.-H. Huang, "A unifying response time analysis framework for dynamic self-suspending tasks," in *Proc. Euromicro Conf. Real Time Syst. (ECRTS)*, 2016, pp. 61–71.

[9] J.-J. Chen *et al.*, "Many suspensions, many problems: A review of self-suspending tasks in real-time systems," *Real Time Syst.*, vol. 55, no. 1, pp. 144–207, 2019.

[10] J.-J. Chen, G. von der Brüggen, W.-H. Huang, and C. Liu, "State of the art for scheduling and analyzing self-suspending sporadic real-time tasks," in *Proc. 23rd IEEE Int. Conf. Embedded Real Time Comput. Syst. Appl. (RTCSA)*, 2017, pp. 1–10.

[11] U. C. Devi, "An improved schedulability test for uniprocessor periodic task systems," in *Proc. 15th Euromicro Conf. Real Time Syst. (ECRTS)*, 2003, pp. 23–32.

[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 *Proc. Real Time Syst. Symp. (RTSS)*, 2016, pp. 339–350.

[13] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *Proc. Workshop Anal. Tools Methodol. Embedded Real Time Syst. (WATERS)*, 2010, pp. 1–6.

[14] M. Günzel and J. Chen, "On schedulability analysis of EDF scheduling by considering suspension as blocking," 2020. [Online]. Available: arXiv.abs/2001.05747.

[15] C. Liu and J. H. Anderson, "Suspension-aware analysis for hard real-time multiprocessor scheduling," in *Proc. 25th Euromicro Conf. Real Time Syst. (ECRTS)*, 2013, pp. 271–281.

[16] C.-L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[17] M. Mohaqeqi, P. Ekberg, and W. Yi, "On fixed-priority schedulability analysis of sporadic tasks with self-suspension," in *Proc. Real Time Netw. Syst. (RTNS)*, 2016, pp. 109–118.

[18] J. C. Palencia and M. G. Harbour, "Response time analysis of EDF distributed real-time systems," *J. Embedded Comput.*, vol. 1, no. 2, pp. 225–237, 2005.

[19] B. Peng and N. Fisher, "Parameter adaption for generalized multiframe tasks and applications to self-suspending tasks," in *Proc. Embedded Real Time Comput. Syst. Appl. (RTCSA)*, 2016, pp. 49–58.

[20] F. Ridouard, P. Richard, and F. Cottet, "Negative results for scheduling independent hard real-time tasks with self-suspensions," in *Proc. Real Time Syst. Symp. (RTSS)*, 2004, pp. 47–56.

[21] G. von der Brüggen, W. Huang, J. Chen, and C. Liu, "Uniprocessor scheduling strategies for self-suspending task systems," in *Proc. ACM Real Time Netw. Syst. (RTNS)*, 2016, pp. 119–128.

[22] G. von der Brüggen, W.-H. Huang, and J.-J. Chen, "Hybrid self-suspension models in real-time embedded systems," in *Proc. Int. Conf. Embedded Real Time Comput. Syst. Appl. (RTCSA)*, 2017, pp. 1–9.