technische universität
dortmund

EDF-Like Scheduling for Self-Suspending Real-Time Tasks

Mario Günzel, Georg von der Brüggen, Kuan-Hsun Chen, Jian-Jia Chen
TU Dortmund University, Department of Computer Science, Dortmund, Germany
University of Twente, Department of Computer Science, preprint.pdf, The Netherlands

Citation:

BIBTEX:

```
@inproceedings{22RTSS_GuenzelBCC,
  author={Mario Günzel, Georg von der Brüggen, Kuan-Hsun Chen, Jian-Jia Chen},
  booktitle={43rd IEEE Real-Time Systems Symposium (RTSS)},
  title={EDF-Like Scheduling for Self-Suspending Real-Time Tasks},
  year={2022},
  volume={},
  number={},
  pages={},
  doi={}
}
```


computer
science 12

EDF-Like Scheduling for Self-Suspending Real-Time Tasks

Mario Günzel*, Georg von der Brüggen*, Kuan-Hsun Chen‡, and Jian-Jia Chen*
*TU Dortmund University, Germany, {mario.guenzel, georg.von-der-brueggen, jian-jia.chen}@tu-dortmund.de
‡University of Twente, The Netherlands, k.h.chen@utwente.nl

***Abstract*—In real-time systems, schedulability analyses provide the required timing guarantees. However, current suspension-aware analyses are limited to Task-Level Fixed-Priority (TFP) scheduling or Earliest-Deadline-First (EDF) scheduling of constrained-deadline self-suspending task systems. In this work, we provide a unifying schedulability analysis for uniprocessor Global EDF-Like (GEL) schedulers of arbitrary-deadline task sets. While analyses for EDF-Like schedulers are rare, many widely used scheduling algorithms can be considered as EDF-Like, for example, EDF, First-In-First-Out (FIFO), Earliest-Quasi-Deadline-First (EQDF), and Suspension-Aware EDF (SAEDF). Therefore, the provided analysis is applicable to those algorithms. It can be applied to TFP scheduling as well. Our analysis is the first suspension-aware schedulability analysis for arbitrary-deadline sporadic real-time task systems under Job-Level Fixed-Priority (JFP) scheduling, such as EDF, and the first unifying suspension-aware schedulability analysis framework that covers a wide range of scheduling algorithms. Through numerical simulations, we show that our analysis improves the state of the art for constrained-deadline EDF scenarios.**

***Index Terms*—Real-Time Systems, Schedulability Analysis, EDF-Like, Self-Suspension**

I. INTRODUCTION

In *real-time systems*, jobs (task instances) of recurrent tasks have to satisfy timing constraints. That is, each job has to finish no later than its absolute deadline. Hence, the compliance to these timing constraints under a given *scheduling algorithm* has to be verified using a related schedulability test. Most classical analysis techniques, like the critical instant theorem [28], Time-Demand Analysis (TDA) [21], [24], or the demand bound function [3], are based on the assumption that a job, after it is released, is either executed or waiting to be executed in the ready queue until it finishes.

Contrarily, a job of a *self-suspending* task may release the processor before being completed, for instance when waiting to get access to a shared resource or offloading computation to an external device, and continue its execution later on. In this setting, the classical critical instant theorem does not hold and related early results [1], [22] have been disproved, cf., [9]. Recently, a large number of additional results have also been reported to be flawed in [9], [14], [15], [31], showing that analyses for self-suspending task systems is non-trivial.

A vast majority of the literature studies either the *segmented* or the *dynamic* self-suspension model. The *segmented* model [5]–[7], [18], [19], [23], [30], [32], [33], [36] predefines an iterating pattern of execution and suspension intervals for the execution behavior of all jobs for each task, based on execution and suspension time upper bounds for each segment. The *dynamic* model [1], [8], [11], [16], [20], [27] assumes that, as long as the maximum suspension time of the related task is not exceeded, jobs may suspend as often and as long as desired. A hybrid self-suspension model was proposed by von der Brüggen et al. [35], which can improve the modelling accuracy of the dynamic self-suspension model and increase the flexibility of the segmented self-suspension model. Detailed discussions can be found in the survey papers by Chen et al. [9], [10]. In this work, we focus on dynamic self-suspending tasks on a single processor, whereas the scheduling algorithm and (sufficient) schedulability test can be used for the segmented or hybrid self-suspension model as well.

Previous work for self-suspending task sets considered either Task-Level *Fixed-Priority* (TFP) scheduling or Earliest-Deadline-First (EDF) scheduling. For preemptive TFP (i.e., if one task has higher priority than another, all its jobs have higher priority as well), the work by Chen [8] (and its recent extension by Günzel et al. [17] to arbitrary-deadline and arrival curves) dominate the other schedulability tests derived in the literature [1], [20], [22], [29]. For task-level dynamic-priority scheduling algorithms (where the priority of the jobs of one task may differ over time) only EDF (where the priority of a job is specified by its absolute deadline) has been studied. The result by Devi [11] has recently been disproved [14]. The results by Liu and Anderson [26] and Dong and Liu [12] for global EDF can be applied for uniprocessor systems by setting the number of processors to one. The only dedicated analysis for EDF on uniprocessor systems by Günzel et al. [16] significantly improves their results in the uniprocessor setting.

In this work, we consider the *window-constrained* scheduler EDF-Like (EL). The category of *window-constrained* schedulers, where at each time job priorities are assigned according to a *priority point (PP)*, has originally been proposed by Leontyev and Anderson [25] to provide general tardiness bounds for multiprocessor scheduling. The popular task-level dynamic-priority algorithms, such as EDF, First-In-First-Out (FIFO), and EQDF [2] fall into this category.

<u>Contributions</u>: We provide the first unifying suspension-aware schedulability test for uniprocessor EDF-Like (EL) scheduling that can be applied to a set of widely used scheduling algorithms, liked EDF, FIFO, EQDF, and TFP, for arbitrary-deadline task systems.

- In Section III, we discuss the capabilities and limitations of EDF-Like (EL) scheduling algorithms and demon-

- strate how they can be configured to behave as EDF, FIFO, EQDF, suspension-aware EDF (SAEDF), TFP algorithms, and hierarchical scheduling.
- In Section IV, we introduce a unifying schedulability test for uniprocessor EL scheduling algorithms, that is applicable to self-suspending arbitrary-deadline task systems. To the best of our knowledge, this is the first result that can handle arbitrary-deadline task sets under Job-Level Fixed-Priority (JFP) scheduling and cover a wide range of scheduling algorithms in one analysis framework for self-suspending task systems.
- Our numerical evaluation in Section V shows that our schedulability test outperforms the state of the art for EDF and performs slightly worse than the test by Chen et al. [8] for Deadline-Monotonic scheduling under constrained-deadlines. We also examine the performance of different configurations for EQDF and SAEDF.

II. System and Scheduling Model

System model. We consider a set $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$ of $n \in \mathbb{N}$ independent self-suspending sporadic real-time tasks in a uniprocessor system. Each task τ_i, $i \in \{1, \ldots, n\}$ is described by a 4-tuple $\tau_i = (C_i, S_i, D_i, T_i)$, composed of worst-case execution time (WCET) $C_i \in [0, D_i]$, maximum suspension time $S_i \geq 0$, relative deadline $D_i \geq 0$, and minimum inter-arrival time $T_i > 0$. It releases an infinite number of jobs according to T_i, i.e., $r_{i,j+1} \geq r_{i,j} + T_i$, where the j^{th} job of τ_i is denoted by $\tau_{i,j}$ and released at time $r_{i,j}$ with $j \in \mathbb{N}$. Each job $\tau_{i,j}$ has to be executed for $c_{i,j} \in [0, C_i]$ time units before its absolute deadline $d_{i,j} = r_{i,j} + D_i$. Each job may suspend itself dynamically, i.e., it may suspend itself as often as desired without exceeding S_i. We denote by $U_i := \frac{C_i}{T_i}$ the utilization of τ_i and by $U := \sum_{i=1}^{n} U_i$ the total utilization of $\mathbb{T}$. A task set $\mathbb{T}$ has *constrained-deadlines* if $D_i \leq T_i$ for every $\tau_i \in \mathbb{T}$; otherwise, it has *arbitrary-deadlines*. We assume continuous time, but our results can be applied for discretized time as well.

Scheduling algorithms. A scheduling algorithm $\mathcal{A}$ specifies the execution behavior of jobs on the processor, that is, it determines at each time, which of the jobs in the ready queue is scheduled. We denote start and finish of job $\tau_{i,j}$ in a certain schedule with $s_{i,j}$ and $f_{i,j}$, respectively, and call a job $\tau_{i,j}$ *finished by* time t, if $f_{i,j} \leq t$. The length of the time interval from release to finish is called the response time $R_{i,j} = f_{i,j} - r_{i,j}$. The *worst-case response time* of a task τ_i is $R_i = \sup_j R_{i,j}$. A schedule is *feasible*, if all jobs finish before or at their absolute deadline, i.e., $f_{i,j} \leq d_{i,j}$ for all $\tau_{i,j}$ or $R_i \leq D_i$ for all τ_i. A task set is *schedulable* by a scheduling algorithm $\mathcal{A}$ if for each job sequence released by $\mathbb{T}$, $\mathcal{A}$ creates a feasible schedule. In this work, we only consider preemptive, work-conserving scheduling, where job execution may be preempted to execute another job, and the processor executes a job whenever there is one in the ready queue.

EDF-Like scheduling. In EDF-Like (EL) scheduling, the priority of each job $\tau_{i,j}$ is based on a job-specific *priority point*

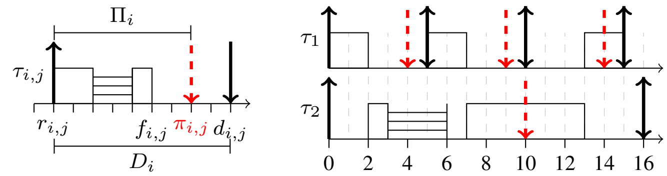


Fig. 1: Left: Presentation of our Notation. Right: Jobs of two tasks scheduled by EL scheduling. The schedule is feasible and the job priority is given by $\pi_{1,1} < \pi_{1,2} < \pi_{2,1} < \pi_{1,3}$.

$$\text{TFP} \subseteq \textbf{EL} \subseteq \text{JFP} \subseteq \text{TDP}$$

Fig. 2: Expressiveness of the scheduling policies Task-Level Fixed-Priority (TFP), EDF-Like (EL), Job-Level Fixed-Priority (JFP), and Task-Level Dynamic-Priority (TDP).

(PP) $\pi_{i,j} \in \mathbb{R}$. More specifically, a job $\tau_{i,j}$ has higher priority than $\tau_{i',j'}$ if $\pi_{i,j} < \pi_{i',j'}$. The priority point is induced by the release of the job and a task specific parameter Π_i denoted as *relative priority point*, i.e., $\pi_{i,j} = r_{i,j} + \Pi_i$. As a result, smaller Π_i in comparison to the other relative priority points favor the jobs of τ_i to be scheduled first. The left hand side of Figure 1 depicts the notation used throughout this work.

Definition 1 (Priority assignment in EL scheduling)**.** Let $\tau_{i,j}$ and $\tau_{i',j'}$ be two different jobs of tasks τ_i and $\tau_{i'}$ in $\mathbb{T}$. Furthermore, let Π_i and $\Pi_{i'}$ be the relative priority points of τ_i and $\tau_{i'}$. The job $\tau_{i,j}$ has *higher priority* than $\tau_{i',j'}$ **if** $\pi_{i,j} < \pi_{i',j'}$ for the priority points $\pi_{i,j} = r_{i,j} + \Pi_i$ and $\pi_{i',j'} = r_{i',j'} + \Pi_{i'}$. If the priority points coincide, i.e., $\pi_{i,j} = \pi_{i',j'}$, then the tie is broken arbitrarily.

Under an assignment of relative priority points $(\Pi_1, \ldots, \Pi_n)$, whenever a job is added to the ready queue, the new highest-priority job is determined and executed. Whenever a job finishes or suspends itself, it is removed from the ready queue and a new highest-priority job is determined and executed.

Example 2. The right hand side of Figure 1 shows a schedule for two tasks $\tau_1 = (C_1 = 2, S_1 = 0, D_1 = 5, T_1 = 5)$ and $\tau_2 = (C_2 = 7, S_2 = 3, D_2 = 16, T_2 = 16)$ under EL scheduling with relative priority points $(\Pi_1 = 4, \Pi_2 = 10)$.

We assume that the jobs of a task τ_i must be executed one after another in a FIFO manner. That is, if a job $\tau_{i,j}$ of task τ_i is only eligible for execution (i.e., ready to be executed) after all jobs of τ_i released prior to $\tau_{i,j}$ finish execution. Specifically, $\tau_{i,j}$ cannot be executed while $\tau_{i,j-1}$ is suspended.

III. Capabilities and Limitations of EL Scheduling

Since the class of EDF-Like (EL) scheduling algorithms is considered sparsely in the literature, in this section, we discuss how they relate to Task-Level Fixed-Priority (TFP), Job-Level Fixed-Priority (JFP), and Task-Level Dynamic-Priority (TDP) scheduling. This relation is depicted in Figure 2.

Since all jobs are assigned priorities based on a fixed priority point, EL scheduling algorithms are by design a subclass of

JFP algorithms, i.e., if one job has higher priority than another job at some point in time, then it has higher priority at all times.[1] While the EL scheduling algorithms are a subset of JFP, it contains many frequently used JFP algorithms:

- Earliest-Deadline-First (EDF) [28] ($\Pi_i = D_i$)
- Earliest-Quasi-Deadline-First (EQDF) [2] ($\Pi_i = D_i + \lambda C_i$ for some predefined $\lambda \in \mathbb{R}$)
- Suspension-aware variations of EDF (SAEDF) ($\Pi_i = D_i + \lambda S_i$ for some predefined $\lambda \in \mathbb{R}$)
- First-In-First-Out (FIFO) ($\Pi_i = 0$)

As a result, the schedulability test that we present in Section IV is applicable to all these scheduling algorithms by configuring the relative priority points Π_i accordingly.

Moreover, Task-Level Fixed-Priority (TFP) algorithms can be treated as EL scheduling algorithms *in the analysis* as well. That is, each TFP algorithm can be transferred into a related EL scheduling algorithm with the same behavior. Hence, if a task set is determined to be schedulable under this EL scheduling algorithm, it is schedulable under the TFP algorithm as well. Assume a given TFP assignment, that the tasks are ordered by their priorities, i.e., τ_k has higher priority than $\tau_{k'}$ if and only if $k < k'$ (with τ_1 having highest priority), and that a worst-case response time (WCRT) upper bound K_j for each task either for the schedule under EL scheduling or under TFP scheduling is given. If we set the relative priority point of each task τ_i to $\Pi_i = \sum_{j=1}^{i} K_j$, then EL and TFP coincide. An EL schedulability test can also be utilized when the relative deadline D_i is taken as an upper bound on the WCRT. In both cases, if a task set is schedulable according to an EL schedulability test, it is also schedulable under the given TFP assignment. For the sake of completeness, we provide a detailed proof for these two statements in the appendix. We note that the response time of tasks may be unbounded and that for such cases a TFP algorithm cannot be treated as EL scheduling algorithm. However, these cases do not apply in practical scenarios if it is required that the jobs of all tasks finish at or before their absolute deadline.

The assignment of relative priority points also allows to mix different scheduling algorithms or hierarchical scheduling algorithms as shown in the following example.

Example 3. We consider a task set $\mathbb{T} = \{\tau_1, \ldots, \tau_4\}$ of 4 tasks. In the following we demonstrate how to assign priorities such that τ_1 and τ_2 are on one priority-level, and τ_3 and τ_4 are on another priority-level, and on each priority-level EDF is utilized. We assign the relative priority points $\Pi_1 = D_1$, $\Pi_2 = D_2$, $\Pi_3 = D_1 + D_2 + D_3$ and $\Pi_4 = D_1 + D_2 + D_4$. If $\mathbb{T}$ is schedulable under EL scheduling with the given relative priority points, then EL produces the same schedule as the desired scheduling policy: Since $\Pi_i - \Pi_j \geq D_i$ for all $i = 3, 4$ and $j = 1, 2$, a job J of τ_1 or τ_2 can only have higher priority than a job J' of τ_3 or τ_4 if J' is already finished when J is released. τ_1 and τ_2 are scheduled according to EDF, since their relative priority points are set to the deadline. The tasks τ_3 and τ_4 are also scheduled according to EDF, since the difference between the global priorities $\pi_{3,j}$ and $\pi_{4,j'}$ of each two jobs $\tau_{3,j}$ and $\tau_{4,j'}$ is the same as the difference between the absolute deadlines $r_{3,j} + D_3$ and $r_{4,j'} + D_4$.

To conclude, the expressiveness of EDF-Like (EL) scheduling algorithms is between Task-Level Fixed-Priority (TFP) and Job-Level Fixed-Priority (JFP) scheduling, but includes many important JFP algorithms like EDF, EQDF, FIFO, and SAEDF. Furthermore, EL scheduling allows to mix different scheduling strategies and hierarchical scheduling.

[1]Please note that Leontyev and Anderson [25] define priority points as well. However, they obtain priority points by prioritization functions, thus their model has the same expressiveness as TDP.

IV. Schedulability Test for EL Scheduling Algorithms

In this section, we derive a sufficient schedulability test for EL scheduling. That is, for an arbitrary-deadline task set $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$ and an assignment of relative priority points $(\Pi_1, \ldots, \Pi_n)$ the test returns *True* if $\mathbb{T}$ is schedulable by the corresponding EL scheduling algorithm.

Our high-level idea is to bound the worst-case response time of a task τ_k by looking at one job $\tau_{k,\ell}$, bounding the time the job needs to run, the time the job can be suspended, and the possible interference, both from higher-priority tasks and from earlier jobs of the same task τ_k.

We summarize the individual steps in the following roadmap, which we depict in Figure 3:

- In Section IV-A, we formalize the above intuition by defining two processor states from the perspective of $\tau_{k,\ell}$, namely the intervals where the processor is working on or suspended by jobs of τ_k and the intervals where the processor is blocked by higher-priority jobs. This leads to our analysis backbone, the response time upper bound in Theorem 9, which assumes that upper bounds for all the interference values are known.
- In Section IV-B, we provide upper bounds for the self-interference, denoted $\sum_{j<\ell} \mathrm{WS}_{k,j}$, and for the interference from other tasks, denoted $B^i_{k,\ell}$, when EL scheduling is applied.
- We show in Section IV-C how the analysis can be conducted for a fixed analysis window, i.e., we only analyze active intervals starting when $\tau_{k,\ell}$ is already released.
- Moreover, we provide an approach with a gradually increasing analysis window in Section IV-D.

Although increasing the analyzed active interval may reduce the worst-case response time bound from the analysis, in Section IV-E we discuss why the two approaches from Section IV-C and Section IV-D do not dominate each other.

A. Examination of Processor States

We first define the terminology to describe the processor state.

Definition 4 (Active and Current Job)**.** For a certain schedule, a job $\tau_{k,\ell}$ of a task τ_k is *active* at time t, if it is released but not already finished by time t, i.e., $t \in [r_{k,\ell}, f_{k,\ell})$. When there are active jobs of task τ_k at a time instant t, then we call the active job of τ_k with the earliest release time the *current* job

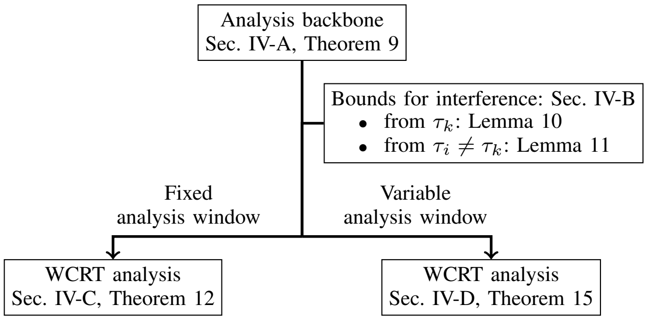


Fig. 3: Roadmap of the analysis in Section IV.

of τ_k at time t. We call the task τ_k active at t, if there exists an active job of τ_k at t.

Definition 5 (Work, Suspend, and Wait)**.**

- The processor is *working on* a job $\tau_{i,j}$ at time t, if $\tau_{i,j}$ is executed on the processor at t. It is *suspended by* $\tau_{i,j}$ at time t, if $\tau_{i,j}$ is suspending itself at t, i.e., the remaining suspension time of $\tau_{i,j}$ is reduced.
- We say that the processor is *working* at time t if it is working on any job at t. It is *suspended* at t, if it is suspended by at least one job but not working on any job at t. It is *waiting* at t, if it is neither working nor suspended at t. The processor is *idle* at t, if it is not working at t, i.e., if it is suspended or waiting.

For unambiguous partition of the processor to the different states, we use half-opened intervals, e.g., if the processor is working on a job $\tau_{i,j}$ from time t_1 to time t_2, then we say that *the processor is working on* $\tau_{i,j}$ *during* $[t_1, t_2)$.

We consider a schedule obtained by the EL scheduling algorithm with relative priority points $(\Pi_1, \dots, \Pi_n)$ for the task set $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$. For each task τ_k we define the following two **Processor States** (PS):

- PS_k^b: There is an active job of τ_k but the processor is working on a job with higher priority than the current job of task τ_k, i.e., τ_k is *blocked*.
- PS_k^{ws}: There is an active job of τ_k and the processor is *working* on or *suspended* by a job of τ_k.

Whenever there is an active job of τ_k and the current job $\tau_{k,\ell}$ of τ_k is not blocked by a higher priority job, then the processor is either working on or suspended by $\tau_{k,\ell}$ since the underlying scheduling algorithm is work-conserving. This leads to the following observation.

Observation 6. *Whenever there is an active job of τ_k, the processor is in state PS_k^b or in state PS_k^{ws}.*

Please note that the processor can also be in both states at the same time, i.e., the processor is suspended by the current job of τ_k and then a higher priority job is released and the processor starts working on the higher priority job. However, it is sufficient for our analysis, that the processor is in *at least* one of the states PS_k^b and PS_k^{ws}.

Under the assumption that the job $\tau_{k,\ell}$ is the current job of τ_k, the time in the two processor states can be described by the following terms.

Definition 7. Let $\tau_{k,\ell}$ be a job of τ_k. Furthermore, let $[c, d)$ with $c < d$ be any half opened interval.

- $\mathrm{B}_{k,\ell}^i(c, d)$ is the amount of time during $[c, d)$ that the processor is working on jobs of τ_i with higher priority than $\tau_{k,\ell}$.
- $\mathrm{WS}_{k,\ell}(c, d)$ is the amount of time during $[c, d)$ that the processor is working on or suspended by $\tau_{k,\ell}$.

If $c \geq d$, we set both terms to 0 for simplicity.

In particular, if $\tau_{k,\ell}$ is current during $[c, d)$, then $\sum_{i \neq k} \mathrm{B}_{k,\ell}^i(c, d)$ is the amount of time during $[c, d)$ that the processor in state PS_k^b and $\mathrm{WS}_{k,\ell}(c, d)$ is the amount of time during $[c, d)$ that the processor in state PS_k^{ws}. Moreover, if a a previous job $\tau_{k,j}$, $j < \ell$ of τ_k is current during $[c, d)$, then $\sum_{i \neq k} \mathrm{B}_{k,\ell}^i(c, d)$ is an *upper bound* on the amount of time during $[c, d)$ that the processor is in state PS_k^b.

We utilize the description of the processor states to derive a necessary condition for $\tau_{k,\ell}$ not being finished.

Lemma 8. *Consider some interval $[c, d)$ with $c \leq d$. If $\tau_{k,\ell}$ is not finished by time d and the task τ_k is active during (the whole interval) $[c, d)$, then*

$$(C_k + S_k) + \sum_{i \neq k} \mathrm{B}_{k,\ell}^i(c, d) + \sum_{j < \ell} \mathrm{WS}_{k,j}(c, d) > (d - c). \quad (1)$$

Proof. By Observation 6, the processor is in state PS_k^b or in state PS_k^{ws} at all times during $[c, d)$.

Let $\xi \geq 0$ be the non-negative integer, such that $\tau_{k,\ell-\xi}, \dots, \tau_{k,\ell}$ are the current jobs of τ_k during the interval $[c, d)$. Moreover, let $\bigcup_{j=\ell-\xi}^{\ell} [c_j, d_j) = [c, d)$ be a partition of $[c, d)$, such that $\tau_{k,j}$ is current during $[c_j, d_j)$.

Since $\sum_{i \neq k} \mathrm{B}_{k,\ell}^i(c_j, d_j)$ and $\mathrm{WS}_{k,j}(c_j, d_j)$ are upper bounds for the amount of time during $[c_j, d_j)$ in state PS_k^b and PS_k^{ws}, respectively, we obtain

$$d_j - c_j \leq \sum_{i \neq k} \mathrm{B}_{k,\ell}^i(c_j, d_j) + \mathrm{WS}_{k,j}(c_j, d_j) \quad (2)$$

for all $j \in \{\ell - \xi, \dots, \ell\}$. Summing up the individual bounds for all j yields

$$d - c \leq \sum_{i \neq k} \mathrm{B}_{k,\ell}^i(c, d) + \sum_{j=\ell-\xi}^{\ell} \mathrm{WS}_{k,j}(c_j, d_j). \quad (3)$$

Since $\sum_{j=\ell-\xi}^{\ell} \mathrm{WS}_{k,j}(c_j, d_j) \leq \sum_{j \leq \ell} \mathrm{WS}_{k,j}(c_j, d_j) \leq \sum_{j \leq \ell} \mathrm{WS}_{k,j}(c, d)$, we obtain

$$d - c \leq \sum_{i \neq k} \mathrm{B}_{k,\ell}^i(c, d) + \sum_{j \leq \ell} \mathrm{WS}_{k,j}(c, d). \quad (4)$$

Moreover, we have $\mathrm{WS}_{k,\ell}(c, d) < C_k + S_k$ because $\tau_{k,\ell}$ is not finished at time d. Applying this to Equation (4) leads to the equation presented in the lemma. $\square$

By contraposition, if Equation (1) does not hold, then d is an upper bound on the finishing time of $\tau_{k,\ell}$. We utilize Lemma 8 to provide a response time bound $\widetilde{R}_{k,\ell} \leq D_k$ for $\tau_{k,\ell}$ by setting $d = r_{k,\ell} + \widetilde{R}_{k,\ell}$. Enlarging the window of interest from $[c, d)$ to $[c, d_{k,\ell})$ enables the following response time bound which serves as the analysis backbone for the remainder of this section.

Theorem 9 (Analysis Backbone)**.** *Let $c \leq d_{k,\ell} \in \mathbb{R}$ such that either 1) $\tau_{k,\ell}$ is released before c, i.e., $c \geq r_{k,\ell}$, or 2) $c < r_{k,\ell}$ and task τ_k is active during $[c, r_{k,\ell})$. If*

$$\widetilde{R}_{k,\ell} := (C_k + S_k) + \sum_{i \neq k} \mathrm{B}^i_{k,\ell}(c, d_{k,\ell}) + \sum_{j<\ell} \mathrm{WS}_{k,j}(c, d_{k,\ell}) + c - r_{k,\ell}, \tag{5}$$

is at most D_k, then $\widetilde{R}_{k,\ell}$ is an upper bound on the response time of $\tau_{k,\ell}$.

Proof. We prove by contradiction and assume that $\widetilde{R}_{k,\ell}$ is not an upper bound on the response time of $\tau_{k,\ell}$, i.e., the job $\tau_{k,\ell}$ is not finished at time $r_{k,\ell} + \widetilde{R}_{k,\ell}$. We set $d := r_{k,\ell} + \widetilde{R}_{k,\ell}$. Lemma 8 can be applied for the following reasons:

- $d = r_{k,\ell} + \widetilde{R}_{k,\ell} = (C_k + S_k) + \sum_{i\neq k} \mathrm{B}^i_{k,\ell}(c, d_{k,\ell}) + \sum_{j<\ell} \mathrm{WS}_{k,j}(c, d_{k,\ell}) + c \geq c$ by the definition of $\widetilde{R}_{k,\ell}$ in Equation (5).
- $\tau_{k,\ell}$ is not finished at time d by assumption.
- Since 1) or 2) from the description of the theorem holds, this means that τ_k is active during the interval $[c, d)$.

Since Lemma 8 can be applied this means that Equation (1) holds. We have $\mathrm{B}^i_{k,\ell}(c, d) \leq \mathrm{B}^i_{k,\ell}(c, d_{k,\ell})$ for all $i \neq k$ and $\mathrm{WS}_{k,j}(c, d) \leq \mathrm{WS}_{k,j}(c, d_{k,\ell})$ for all $j < \ell$ since $d \leq d_{k,\ell}$. Hence, the left hand side of Equation (1) is upper bounded by $\widetilde{R}_{k,\ell} - c + r_{k,\ell}$. We conclude that $\widetilde{R}_{k,\ell} - c + r_{k,\ell} > d - c = \widetilde{R}_{k,\ell} + r_{k,\ell} - c$ which is a contradiction. □

So far, Theorem 9 examines a given interval that starts at time c and ends at time $d_{k,\ell}$, i.e., the absolute deadline of the analysed job, under the assumption that the inference in the interval is known. However, how to choose the starting value c and how the interference in $[c, d_{k,\ell})$ can actually be calculated has not yet been discussed. Hence, to apply the upper bound in Theorem 9, the following questions have to be answered:

- **Question 1:** What are the values of $\mathrm{B}^i_{k,\ell}(c, d_{k,\ell})$ and $\mathrm{WS}_{k,j}(c, d_{k,\ell})$? Since computing the values directly has high complexity, we use overapproximation. In Section IV-B we derive upper bounds for $\mathrm{B}^i_{k,\ell}(c, d_{k,\ell})$ and $\mathrm{WS}_{k,j}(c, d_{k,\ell})$ with $i \neq k$ and $j < \ell$.
- **Question 2:** Which are good values for c? Trying out all possible c for the estimation would result in very high complexity. Therefore, we discuss two strategies to choose c in Sections IV-C and IV-D. More precisely, with the first procedure we restrict c to be in the interval $[r_{k,\ell}, d_{k,\ell})$, which has benefits on the runtime of our analysis due to the *fixed analysis windows*. For the second strategy, we examine *active intervals* for τ_k, and gradually increase the analysis window. In Section IV-E we discuss that these two methods do not dominate each other.

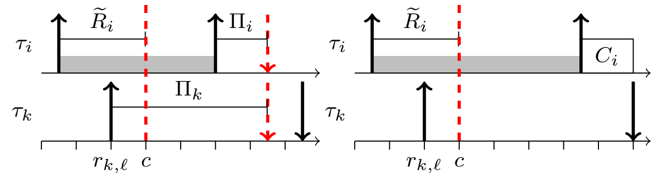


Fig. 4: Intuition for Lemma 11. The left schedule describes the approach for Equation (8) and the right schedule describes the approach for Equation (9). Only jobs of τ_i that are released during the gray boxes can have higher priority than $\tau_{k,\ell}$ and be executed during the analysis interval $[c, d_{k,\ell})$.

B. Upper Bounds on Higher-Priority Interference

In this subsection, we bound the interference of higher-priority jobs during the interval $[c, d_{k,\ell})$, providing upper bounds for $\sum_{j<\ell} \mathrm{WS}_{k,j}(c, d_{k,\ell})$ in Lemma 10 and for $\mathrm{B}^i_{k,\ell}(c, d_{k,\ell})$ in Lemma 11. We do this under the assumption that all jobs with higher priority than $\tau_{k,\ell}$ meet their deadline since this is our induction hypothesis later used in Theorem 12 and Lemma 14. As mentioned earlier, how to choose c is discussed in Sections IV-C and IV-D.

For arbitrary deadlines, there might be several active jobs of one task at the same time, which makes the analysis in general more complicated. Note that, according to the FIFO mechanism introduced in the end of Section II, the jobs of τ_k must be executed one after another, i.e., even if the processor idles, a job of τ_k cannot start its execution unless all jobs of τ_k released prior to it are finished.

We achieve a bound for $\sum_{j<\ell} \mathrm{WS}_{k,j}(c, d_{k,\ell})$ by accounting for the WCET C_k and the maximal suspension time S_k for all higher priority jobs current during $[c, d_{k,\ell})$.

Lemma 10 (Bound for interference from τ_k)**.** *The amount of time during $[c, d_{k,\ell})$ that the processor is working on or suspended for jobs of τ_k with by higher priority than $\tau_{k,\ell}$ is*

$$\sum_{j<\ell} \mathrm{WS}_{k,j}(c, d_{k,\ell}) \leq \left(\left\lceil \frac{d_{k,\ell} - c}{T_k} \right\rceil - 1 \right) \cdot (C_k + S_k). \tag{6}$$

Proof. All higher-priority jobs of τ_k finish until their deadline. Therefore, the processor can only work on or be suspended by a higher priority job of τ_k during $[c, d_{k,\ell})$ if the deadline of the job is after c. Moreover, the job can only have higher priority than $\tau_{k,\ell}$ if its deadline is no later than $d_{k,\ell} - T_k$. At most $\left\lceil \frac{d_{k,\ell}-c}{T_k} \right\rceil - 1$ jobs of τ_k have their deadline during $(c, d_{k,\ell} - T_k]$. The processor can work on each of those jobs for at most C_k time units and it can be suspended by each of those jobs for at most S_k time units. □

For the interference from $\tau_i \neq \tau_k$, we estimate the number of job releases during a certain interval under analysis, as depicted by the gray boxes in Figure 4, and account for C_i time units for each of those jobs.

For each task $\tau_i \neq \tau_k$, let $\widetilde{R}_i$ be an upper bound on the worst-case response time (WCRT) of jobs of τ_i with higher priority than $\tau_{k,\ell}$. We set $\widetilde{R}_i := D_i$ if no upper bound is known, as all jobs with higher priority meet their deadline.

Lemma 11 (Bound for interference from $\tau_i \neq \tau_k$)**.** *For $i \neq k$, the amount of time during $[c, d_{k,\ell})$ that the processor is working on jobs of τ_i with higher priority than $\tau_{k,\ell}$ is*

$$\mathrm{B}^i_{k,\ell}(c, d_{k,\ell}) \leq \max\left(\left\lceil \frac{G^i_k + \widetilde{R}_i + r_{k,\ell} - c}{T_i} \right\rceil, 0\right) C_i \tag{7}$$

where $G^i_k := \min(D_k - C_i, \Pi_k - \Pi_i)$.

Proof. The bound from Equation (7) is achieved by proving the two upper bounds for $\mathrm{B}^i_{k,\ell}$ we get for both cases of G^i_k. That is, we prove the following two bounds individually:

$$\mathrm{B}^i_{k,\ell}(c, d_{k,\ell}) \leq \max\left(\left\lceil \frac{\Pi_k - \Pi_i + \widetilde{R}_i + r_{k,\ell} - c}{T_i} \right\rceil, 0\right) C_i \tag{8}$$

$$\mathrm{B}^i_{k,\ell}(c, d_{k,\ell}) \leq \max\left(\left\lceil \frac{D_k - C_i + \widetilde{R}_i + r_{k,\ell} - c}{T_i} \right\rceil, 0\right) C_i \tag{9}$$

Bound in Equation (8)**:** The bound is based on the following two observations:

- Jobs of τ_i have higher priority than $\tau_{k,\ell}$ only if they are released no later than $r_{k,\ell} + \Pi_k - \Pi_i$.
- Jobs of τ_i can be executed after c only if they are released after $c - \widetilde{R}_i$.

Due to these two observations, the number of jobs that contribute to $\mathrm{B}^i_{k,\ell}(c, d_{k,\ell})$ is upper bounded by the number of releases in $(c - \widetilde{R}_i, r_{k,\ell} + \Pi_k - \Pi_i]$, which is at most $\max\left(\left\lceil \frac{\Pi_k - \Pi_i + \widetilde{R}_i + r_{k,\ell} - c}{T_i} \right\rceil, 0\right)$. The processor can work on each of them for at most C_i time units.

Bound in Equation (9)**:** Before formally proving Equation (9), we first examine the worst-case scenario depicted in Figure 4. Intuitively, the maximal interference from task τ_i is obtained when the *last* interfering job of τ_i is released at $d_{k,\ell} - C_i$ and executed for C_i time units during $[d_{k,\ell} - C_i, d_{k,\ell})$ as depicted in Figure 4. The maximum number of jobs that contribute to $\mathrm{B}^i_{k,\ell}(c, d_{k,\ell})$ is therefore upper bounded by the number of releases in $(c - \widetilde{R}_i, d_{k,\ell} - C_i]$, which is at most $\max\left(\left\lceil \frac{D_k - C_i + \widetilde{R}_i + r_{k,\ell} - c}{T_i} \right\rceil, 0\right)$. The processor can work on each of them for at most C_i time units.

In the following we present a formal proof for the bound from Equation (9). If the processor is not working on any job of τ_i during $[c, d_{k,\ell})$ then $\mathrm{B}^i_{k,\ell}(c, d_{k,\ell}) = 0$ and the lemma is proven. Otherwise, let $\tau_{i,j'}$ be the last job of τ_i that the processor is working on during $[c, d_{k,\ell})$. We isolate the job $\tau_{i,j'}$ in the following way. Let $r_{i,j'}$ be the release time of $\tau_{i,j'}$, let $s_{i,j'}$ be the start time of $\tau_{i,j'}$, i.e., the first time that the processor is working on $\tau_{i,j'}$, and let C^* be the amount of time that the processor is working on $\tau_{i,j'}$ during the interval $[c, d_{k,\ell})$. Therefore, by definition, we have

$$r_{i,j'} + C^* \leq d_{k,\ell} = r_{k,\ell} + D_k \tag{10}$$

and

$$\mathrm{B}^i_{k,\ell}(c, d_{k,\ell}) \leq \mathrm{B}^i_{k,\ell}(c, s_{i,j'}) + C^*. \tag{11}$$

We distinguish two cases:

Case 1: $s_{i,j'} - c < (C_i - C^*)$: The left hand side of Equation (9) is $\mathrm{B}^i_{k,\ell}(c, s_{i,j'}) + C^* \leq (s_{i,j'} - c) + C^* \leq C_i$. Moreover, the right hand side of Equation (9) is at least $\left\lceil \frac{D_k - C_i + r_{k,\ell} - c + \widetilde{R}_i}{T_i} \right\rceil C_i \geq \left\lceil \frac{D_k + r_{k,\ell} - c}{T_i} \right\rceil C_i \geq C_i$ since:

- $\widetilde{R}_i \geq C_i$ by definition of $\widetilde{R}_i$.
- $\tau_{i,j'}$ is executed during $[c, d_{k,\ell})$ and therefore $c < c_{k,\ell} \leq D_k + r_{k,\ell}$ must hold.

In this case, Equation (9) is proven.

Case 2: $s_{i,j'} - c \geq (C_i - C^*)$: Since during the interval $[c, c + (C_i - C^*))$ the processor can work on jobs of τ_i for at most $(C_i - C^*)$ time units, we have $\mathrm{B}^i_{k,\ell}(c, s_{i,j'}) \leq (C_i - C^*) + \mathrm{B}^i_{k,\ell}(c + (C_i - C^*), s_{i,j'})$. Hence, we obtain:

$$
\begin{aligned}
& \mathrm{B}^i_{k,\ell}(c, d_{k,\ell}) && (12)\\
\overset{\text{by (11)}}{\leq}\ & \mathrm{B}^i_{k,\ell}(c, s_{i,j'}) + C^* && (13)\\
\leq\ & (C_i - C^*_i) + \mathrm{B}^i_{k,\ell}(c + (C_i - C^*), s_{i,j'}) + C^*_i && (14)\\
=\ & \mathrm{B}^i_{k,\ell}(c + (C_i - C^*), s_{i,j'}) + C_i && (15)
\end{aligned}
$$

The number of jobs of τ_i that contribute to $\mathrm{B}^i_{k,\ell}(c + (C_i - C^*), s_{i,j'})$ is at most the number of releases from jobs of τ_i during the interval $(c + C_i - C^* - \widetilde{R}_i, r_{i,j'} - T_i]$, which is

$$
\begin{aligned}
& \left\lceil \frac{r_{i,j'} - T_i - c - C_i + C^* + \widetilde{R}_i}{T_i} \right\rceil \\
\overset{\text{by (10)}}{\leq}\ & \left\lceil \frac{r_{k,\ell} - T_i - c - C_i + D_k + \widetilde{R}_i}{T_i} \right\rceil \\
=\ & \left\lceil \frac{r_{k,\ell} - c - C_i + D_k + \widetilde{R}_i}{T_i} \right\rceil - 1.
\end{aligned}
$$

Therefore, for this case, we conclude that

$$
\begin{aligned}
\mathrm{B}^i_{k,\ell}(c, d_{k,\ell}) \leq\ & \mathrm{B}^i_{k,\ell}(c + (C_i - C^*), s_{i,j'}) + C_i \\
\leq & \left(\left\lceil \frac{r_{k,\ell} - c - C_i + D_k + \widetilde{R}_i}{T_i} \right\rceil - 1\right) C_i + C_i \\
= & \left\lceil \frac{r_{k,\ell} - c - C_i + D_k + \widetilde{R}_i}{T_i} \right\rceil C_i.
\end{aligned}
$$

Hence, in this case Equation (9) is proven as well. □

C. Fixed Analysis Window

In this subsection, we fix the analysis window, i.e., the possible range of $[c, d_{k,\ell})$ from the previous sections, to the interval $[r_{k,\ell}, d_{k,\ell})$. We utilize the upper bounds on $\mathrm{B}^i_{k,\ell}(c, d_{k,\ell})$ and $\mathrm{WS}_{k,j}(c, d_{k,\ell})$ provided in the previous section to obtain the following schedulability test.

Algorithm 1 Schedulability test with fixed analysis window.

Input: $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$, $(\Pi_1, \ldots, \Pi_n)$, η, *depth*
Output: True: schedulable, False: no decision

Order $\tau_1, \ldots, \tau_n$, s.th. $D_1 \geq \cdots \geq D_n$.
Set $\widetilde{R}_i := D_i$ for all i.
for $i = 1, 2, \ldots, depth$ **do**
  $solved := True$
  for $k = 1, 2, \ldots, n$ **do**
    $cand := [\,]$; $step := \eta \cdot D_k$ ▷ Preparation.
    for $b = 0, step, 2 \cdot step, \cdots < D_k$ **do** ▷ Compute.
      $cand.append(\widetilde{R}_k(b))$ using Equation (16).
    $\widetilde{R}_k := \min(cand)$ ▷ Compare candidates.
    if $\widetilde{R}_k > D_k$ **then** ▷ Check condition.
      $solved := False$; $\widetilde{R}_k := D_k$; **break**
return $solved$

Theorem 12 (Sufficient Schedulability Test)**.** *Let $\mathbb{T} = \{\tau_1, \ldots \tau_n\}$ be an arbitrary-deadline task set with relative priority points $(\Pi_1, \ldots, \Pi_n)$. If for all $k = 1, \ldots, n$ there exists some $b_k \in [0, D_k)$ such that*

$$\widetilde{R}_k(b_k) \leq D_k, \tag{16}$$

where $\widetilde{R}_k(b_k) := \sum_{i \neq k} \max\left(\left\lceil \frac{G_k^i + \widetilde{R}_i - b_k}{T_i} \right\rceil, 0\right) C_i + \left\lceil \frac{D_k - b_k}{T_k} \right\rceil (C_k + S_k) + b_k$ *and* $G_k^i = \min(D_k - C_i, \Pi_k - \Pi_i)$*, then the task set is schedulable by EL scheduling with the given relative priority points and the worst-case response time of τ_k is upper bounded by $\widetilde{R}_k := \widetilde{R}_k(b_k)$.*

Proof. Assume we have found b_k, $k = 1, \ldots, n$ such that Equation (16) holds. We consider some schedule obtained by the task set $\mathbb{T}$ and denote by Seq the sequence of all jobs in the schedule ordered by their priority. Via induction, we prove that the first ξ jobs in Seq have the required response time upper bound, for all $\xi \in \mathbb{N}_0$. Consequently, $\widetilde{R}_k$ is an upper bound on the worst-case response time of τ_k for all k and the task set is schedulable.

Initial case: $\xi = 0$. In this case, the set of the first ξ jobs in Seq is the empty set. Trivially, all of them have the required response time upper bound.

Induction step: $\xi \to \xi + 1$. By assumption, the first ξ jobs in Seq have the required response time upper bound. We denote the $(\xi + 1)$-th job in Seq by $\tau_{k,\ell}$. We aim to use the analysis backbone from Theorem 9 to prove that the response time of $\tau_{k,\ell}$ is upper bounded by $\widetilde{R}_k$. By definition, we have $\widetilde{R}_{k,\ell} = (C_k + S_k) + \sum_{i \neq k} \mathrm{B}_{k,\ell}^i(c, d_{k,\ell}) + \sum_{j < \ell} \mathrm{WS}_{k,j}(c, d_{k,\ell}) + c - r_{k,\ell}$. Since all higher priority jobs have the required response time upper bound, we can use the estimation for $\mathrm{B}_{k,\ell}^i(c, d_{k,\ell})$ and $\sum_{j<\ell} \mathrm{WS}_{k,j}(c, d_{k,\ell})$ presented in Section IV-B. In particular, using Lemma 10 and Lemma 11, we obtain that $\widetilde{R}_{k,\ell}$ is upper bounded by

$$(C_k + S_k) + \sum_{i \neq k} \max\left(\left\lceil \frac{G_k^i + \widetilde{R}_i + r_{k,\ell} - c}{T_i} \right\rceil, 0\right) C_i + \left(\left\lceil \frac{d_{k,\ell} - c}{T_k} \right\rceil - 1\right) \cdot (C_k + S_k) + c - r_{k,\ell}. \tag{17}$$

When choosing $c := b_k + r_{k,\ell}$ we obtain that $\widetilde{R}_{k,\ell}$ is upper bounded by $\widetilde{R}_k$. Due to Equation (16), we know $\widetilde{R}_{k,\ell} \leq \widetilde{R}_k \leq D_k$, i.e., the job meets its deadline. We use Theorem 9 to conclude that $\widetilde{R}_{k,\ell}$ is an upper bound on the response time of $\tau_{k,\ell}$ and therefore $\widetilde{R}_k$ is an upper bound on the response time of $\tau_{k,\ell}$ as well. Since all jobs meet their deadline, the task set is schedulable. □

Although Theorem 12 looks like a classical mechanism extended from time demand analysis (TDA) [21], [24], implementing an efficient schedulability test based on it requires some efforts since the values of $\widetilde{R}_k$ for every task τ_k are dependent on each other. To apply this schedulability test, two critical points have to be addressed:

1) Finding good values for b_k with low complexity. Without an efficient mechanism, there are D_k options for b_k, provided that all input parameters are integers, and infinitely many options in general.
2) Computing the dependent values of $\widetilde{R}_k$ for every task τ_k correctly and efficiently.

To determine the values of b_k, we take a user-specified parameter η and discretize the search space into $\frac{1}{\eta}$ values with a step size $\eta \cdot D_k$. To determine $\widetilde{R}_k$, we go through the task set several times and compute upper bounds for the values of $\widetilde{R}_k$ in each iteration. Improving this search algorithm is out of scope for this paper but may be discussed in future work.

The search algorithm is depicted in pseudocode in Algorithm 1. It takes as input the task set $\mathbb{T}$, the relative priority points $(\Pi_1, \ldots, \Pi_n)$, a step size parameter $\eta \in (0, 1]$ and *depth* to indicate the number improving runs of the search algorithm. It returns *True* if the task set is schedulable by EL scheduling with the given relative priority points. We start by setting $\widetilde{R}_k = D_k$ for all $k = 1, \ldots, n$, and go *depth*-times through the task set ordered by the relative deadline, as we obtained the best results with this ordering. With a step size of $step = \eta \cdot D_k$, i.e., a certain share of D_k like 1 percent, we compute $\widetilde{R}_k(b_k)$ for $b_k = 0, 1 \cdot step, 2 \cdot step, \ldots$ until $b \geq D_k$ is reached. We then take the minimal value of all these candidates and define it as the new $\widetilde{R}_k$. The time complexity of Algorithm 1 is $\mathcal{O}\left(\frac{depth \cdot n^2}{\eta}\right)$.

Please note that the computed values of $\widetilde{R}_k$ are in fact only upper bounds of $\widetilde{R}_k$ from Theorem 12. A reduction of $\widetilde{R}_i$, $i \neq k$ in subsequent iterations reduces the actual value of $\widetilde{R}_k$ as well, since $\widetilde{R}_k$ is monotonically increasing with respect to $\widetilde{R}_i$, for all $i \neq k$.

D. Variable Analysis Window

In this subsection, we detail an approach based on *active intervals*. More specifically, if all jobs finish until the next job release is reached, i.e., $R_k \leq T_k$, then no previous jobs contribute interference to the job under analysis and they can be safely removed from the computation of the worst-case response-time upper bound. However, if $R_k > T_k$ then interference from previous jobs has to be considered. We utilize that a job $\tau_{k,\ell-a}$ can only interfere with $\tau_{k,\ell}$, if τ_k

is active during $[r_{k,\ell-a}, r_{k,\ell})$. For the schedulability test with variable analysis window, we gradually increase the length of the active interval (i.e., $a = 0, 1, 2, \ldots$) and analyze the window $[c, d_{k,\ell})$ with $c \in [r_{k,\ell-a}, d_{k,\ell})$.

With this approach, the pessimism of the interference estimation from higher-priority jobs of the same task is reduced in some cases. Please note that this approach only differs from the approach with a fixed analysis window when considering arbitrary-deadline tasks. For constrained-deadline task sets, the variable analysis window approach coincides with the fixed analysis window approach, as the algorithm stops at $a = 0$ without enlarging the analysis window.[2]

We start by formally defining active intervals.

Definition 13. Let $a \in \mathbb{N}_0$. A job $\tau_{k,\ell}$ is the $(a+1)$-th job in an *active interval* of τ_k, if the following two conditions hold.

- τ_k is active during $[r_{k,\ell-a}, f_{k,\ell})$.
- At time $r_{k,\ell-a}$ there is no active job which is released *before* $r_{k,\ell-a}$.

If $\tau_{k,\ell}$ is the $(a+1)$-th job in an active interval of τ_k, then only $\tau_{k,\ell-a}, \ldots, \tau_{k,\ell}$ are current jobs of τ_k during $[r_{k,\ell-a}, f_{k,\ell})$. More specifically, in this case the value of $B_{k,j}(c, d_{k,\ell})$ is 0 if $c \geq r_{k,\ell-a}$ and $j < \ell - a$. We formalize this by the following lemma.

Lemma 14. *Let $\tau_{k,\ell}$ be the $(a+1)$-th job in an active interval of τ_k and let all higher-priority jobs meet their deadline. Let $\widetilde{R}_i$ be an upper bound on the response time of all higher-priority jobs of τ_i with $i \neq k$. If there exists some $c \in [r_{k,\ell-a}, d_{k,\ell})$ such that*

$$\min\left(a+1, \left\lceil \frac{d_{k,\ell} - c}{T_k} \right\rceil\right)(C_k + S_k) + \sum_{i \neq k} \max\left(\left\lceil \frac{G_k^i + \widetilde{R}_i + r_{k,\ell} - c}{T_i} \right\rceil, 0\right) C_i + c - r_{k,\ell} \tag{18}$$

is at most D_k, with $G_k^i := \min(D_k - C_i, \Pi_k - \Pi_i)$, then (18) *is an upper bound on the response time of $\tau_{k,\ell}$.*

Proof. For the proof, we apply the analysis backbone from Theorem 9. Since $\tau_{k,\ell}$ is the $(a+1)$-th job in an active interval, τ_k is active during $[r_{k,\ell-a}, r_{k,\ell})$. Hence, the restriction on c in the formulation of Theorem 9 is fulfilled if c is chosen from the interval $[r_{k,\ell-a}, d_{k,\ell})$. Moreover, since $\tau_{k,\ell}$ is the $(a+1)$-th job in an active interval, the jobs $\tau_{k,1}, \ldots, \tau_{k,\ell-a-1}$ are finished by time $r_{k,\ell-a}$. We obtain $\sum_{j<\ell-a} \mathrm{WS}_{k,j}(c, d_{k,\ell}) \leq \sum_{j<\ell-a} \mathrm{WS}_{k,j}(r_{k,\ell-a}, d_{k,\ell}) = 0$. Hence, $\sum_{j<\ell} \mathrm{WS}_{k,j}(c, d_{k,\ell}) = \sum_{j=\ell-a}^{\ell-1} \mathrm{WS}_{k,j}(c, d_{k,\ell}) \leq \sum_{j=\ell-a}^{\ell-1} (C_k + S_k) = a \cdot (C_k + S_k)$. We combine this upper bound on $\sum_{j<\ell} \mathrm{WS}_{k,j}(c, d_{k,\ell})$ with the upper bounds from Lemma 10 and Lemma 11, and obtain that $\widetilde{R}_{k,\ell}$ from the analysis backbone, i.e., Equation (5), is upper bounded by the value in Equation (18). If Equation (18) is at most D_k, then $\widetilde{R}_{k,\ell} \leq D_k$. The analysis backbone from Theorem 9 states that $\widetilde{R}_{k,\ell}$ is an upper bound on the response time of $r_{k,\ell}$ and therefore, also Equation (18) is an upper bound on the response time. $\square$

[2]The reason is that for constrained deadline task sets in the variable analysis window approach we get one of the following two cases: Case 1: If the response time bound obtained for the first job in an active interval is larger than the task's deadline, then the schedulability cannot be guaranteed and the algorithm stops at $a = 0$. Case 2: If the response time bound for the first job in an active interval is at most the task's deadline, then $R_k \leq D_k \leq T_k$ is guaranteed for the first job in an active interval. Therefore, the subsequent job is a first job in some active interval as well. By induction, all jobs of τ_k are first job in some active interval. Therefore, the algorithm stops at $a = 0$.

Similar to the proof of the response time upper bound for the fixed analysis window in Theorem 12, we derive a response time bound for the variable analysis window. However, the different cases for $a = 0, 1, \ldots$ have to be considered for each task τ_k and the value of c can be chosen from the interval $[r_{k,l} - aT_k, r_{k,l} + D_k)$. To improve readability, in the following we replace $r_{k,\ell} - c$ by $aT_k - x$, where x can be chosen from the interval $[0, aT_k + D_k)$.

Theorem 15 (Sufficient Schedulability Test)**.** *Let $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$ be an arbitrary-deadline task set with relative priority points $(\Pi_1, \ldots, \Pi_n)$. We define the function $\widetilde{R}_k^a : [0, aT_k + D_k) \to \mathbb{R}_{\geq 0}$ by the assignment*

$$x \mapsto \min\left(a+1, \left\lceil \frac{D_k - x + aT_k}{T_k} \right\rceil\right)(C_k + S_k) + \sum_{i \neq k} \max\left(\left\lceil \frac{G_i^k + \widetilde{R}_i - x + aT_k}{T_i} \right\rceil, 0\right) C_i + x - aT_k. \tag{19}$$

If for all $k = 1, \ldots, n$ there exists $\tilde{a}_k \in \mathbb{N}_0$, such that for all $a = 0, \ldots, \tilde{a}_k$ there exists $b_k^a \in [0, aT_k + D_k)$, such that

$$\widetilde{R}_k^a(b_k^a) \leq D_k, \text{ and furthermore } \widetilde{R}_k^{\tilde{a}_k}(b_k^{\tilde{a}_k}) \leq T_k, \tag{20}$$

then the task set is schedulable by EL scheduling with the given relative priority points and $\widetilde{R}_k := \max_{a=0,\ldots,\tilde{a}} \widetilde{R}_k^a(b_k^a)$ is an upper bound on the WCRT of τ_k for all k.

Proof. The proof is similar to the one of the sufficient schedulability test for the fixed analysis window presented in Theorem 12. Let Seq be the sequence of all jobs in the schedule ordered by their priority. By induction we show that the following response time upper bounds holds for the first $\xi \in \mathbb{N}_0$ jobs in Seq:

(i) $\widetilde{R}_k$ is a response time upper bound for all jobs of τ_k.
(ii) T_k is a response time upper bound for all $\tilde{a}_k$-th jobs in an active interval of τ_k.

Initial case: ξ_0. The initial case is again trivially fulfilled, since there has nothing to be checked when there are no jobs.

Induction step: $\xi \to \xi + 1$. The first ξ jobs in Seq have the required response time upper bounds (i) and (ii) by induction. We denote by $\tau_{k,\ell}$ the $(\xi+1)$-th job of Seq. Let a be the lowest value in $\mathbb{N}_0$, such that $\tau_{k,\ell}$ is the $(a+1)$-th job in an active interval of τ_k.

We first show that $a \leq \tilde{a}_k$ by contraposition. In this regard, we assume $a > \tilde{a}_k$ and consider the job $\tau_{k,\ell-(a-\tilde{a}_k)}$. The job $\tau_{k,\ell-(a-\tilde{a}_k)}$ is the $(\tilde{a}_k + 1)$-th job in an active interval of τ_k. Moreover, this job is one of the first ξ jobs in Seq and therefore

Algorithm 2 Schedulability test with var. analysis window.

```
Input: 𝕋 = {τ_1, ..., τ_n}, (Π_1, ..., Π_n), η, max_a, depth
Output: True: schedulable, False: no decision

Order τ_1, ..., τ_n, s.th. D_1 ≥ ··· ≥ D_n.
Set R̃_i := D_i for all i.
for i = 1, 2, ..., depth do
    solved := True
    for k = 1, 2, ..., n do
        for a = 0, 1, ..., max_a do                          ▷ Different a.
            cand := [ ]; step := η · D_k                     ▷ Preparation.
            for b = 0, step, 2 · step, ··· < aT_k + D_k do   ▷ Compute candidate.
                cand.append(R̃_k^a(b)) from Equation (19)
            R̃_k^a := min(cand)                               ▷ Compare candidates.
            if R̃_k^a ≤ T_k then                              ▷ Check cond. 1.
                ã := a; R̃_k := min_{a=0,...,ã} R̃_k^a; break  ▷ WCRT upper bound.
            if R̃_k^a > D_k or a = max_a then                 ▷ Check cond. 2.
                solved := False; R̃_k := D_k; break
return solved
```

has a response time of at most T_k due to (ii). Hence, the job $\tau_{k,\ell-(a-\tilde{a}_k)}$ is finished by time $r_{k,\ell-(a-\tilde{a}_k)+1}$. We conclude that $\tau_{k,\ell}$ is the $(a - \tilde{a}_k)$-th job in an active interval of τ_k, which contradicts the minimality of a.

We now choose $c := b_k^a - a \cdot T_k + r_{k,\ell}$. Applying the response time upper bound provided by Equation (18) in Lemma 14 with this c shows that $\widetilde{R}_k^a(b_k^a)$ is a response time upper bound of $\tau_{k,\ell}$, which shows that (i) holds for $\tau_{k,\ell}$. Moreover, if $a = \tilde{a}_k$ then we have already shown that $\widetilde{R}_k^{\tilde{a}_k}(b_k^{\tilde{a}_k})$ is an upper bound on the response time of $\tau_{k,\ell}$. By Equation (20) the response time of $\tau_{k,\ell}$ is upper bounded by T_k, which shows that (ii) holds for $\tau_{k,\ell}$ as well. □

We apply a similar search strategy as for the case with fixed analysis window. However, the values of $\widetilde{R}_k$ are computed through an additional loop over the values of a until $\widetilde{R}_k^a \leq T_k$. Algorithm 2 depicts an implementation of the schedulability test in pseudocode. As the value of $\widetilde{R}_k^a$ can be between T_k and D_k for all iterations of a, the program may never return a result. To make the schedulability test deterministic, we introduce an additional parameter max_a which aborts the loop when no result is obtained for $a = 0, 1, ..., max\_a$.

E. Dominance of Fixed and Variable Analysis Window

At first glance, the analysis with variable window size derived in Section IV-D seems to improve the analysis with fixed window size from Section IV-C in all cases: When setting $x = b_k + a \cdot T_k$ in Theorem 15, then the result is lower bounded by $\widetilde{R}_k(b_k)$ from Theorem 12. However, both methods do not dominate each other, as demonstrated in the discussion of Figure 7 in Section V, due to the following reasons. First, the analysis with variable analysis window can only analyze schedules where the length of active intervals is bounded. More specifically, if the response-time upper bound $\widetilde{R}_k^a$ is in the interval (T_k, D_k) for all a, then the schedulability test with the variable analysis window never deems the task schedulable. Second, by setting max_a this effect is even intensified: The analysis with variable analysis window has to find $\widetilde{R}_k^a \leq T_k$ even for some $a \leq max\_a$. Third, the discretization using η in Algorithm 1 and Algorithm 2 ensures the same number of points for each analysis interval. As a result, not all points $b + aT_k$ with b from Algorithm 1 are checked during Algorithm 2 as well.

V. Evaluation

In this section, we evaluate the performance of our schedulability tests **(EL)** presented in Algorithm 1 for the fixed analysis window and in Algorithm 2 for the variable analysis window. More precisely we show that:

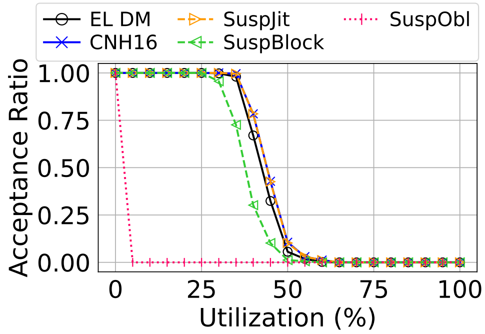
1) Our schedulability test performs similar to already existing schedulability tests for Deadline-Monotonic (DM) and improves the state of the art for Earliest-Deadline-First (EDF) scheduling.
2) Our schedulability test can be used to compare different configurations of Earliest-Quasi-Deadline-First (EQDF) and suspension-aware EDF (SAEDF) (see Section III).
3) Our schedulability test exploits the optimism introduced when the deadline of tasks is extended over their minimum inter-arrival time.

Please note that for constrained deadlines we do not distinguish between fixed and variable analysis window since both schedulability tests coincide, as explained in Section IV-D. When applying our schedulability test, we choose the configuration $\eta = 0.01$, $depth = 5$, and $max\_a = 10$. In each figure, we present the *acceptance ratio*, which is the share of task sets that are deemed schedulable by the schedulability test under consideration. The evaluation is released on Github [34].

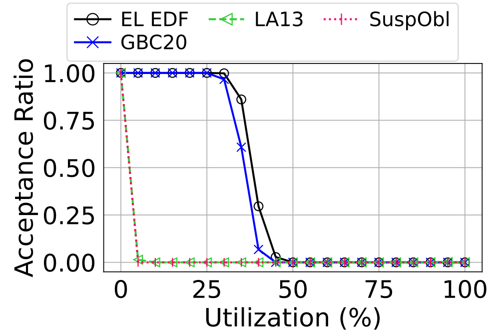
For the evaluation, we synthesized 500 task sets with 50 tasks each for each total system utilization from 0% to 100% in steps of 5%. We first generated 50 utilization values U_i using the UUniFast [4] method with the given total utilization goal, and adopted the suggestion by Emberson et al. [13] to draw the minimum inter-arrival time T_i according to a log-uniform distribution from the interval $[1, 100][ms]$. The worst-case execution time was computed as $C_i = T_i \cdot U_i$ and the deadline was set to the minimum inter-arrival time, i.e., $D_i = T_i$. For each task, we drew the maximum suspension time S_i uniformly at random from $[0, 0.5(T_i - C_i)]$. The tasks in each set were ordered by their deadline.

In Figure 5a, we show the results when applying EL with relative priority points $\Pi_i = \sum_{j=1}^{i} D_j$ to obtain a schedulability test for DM scheduling **(EL DM)**. We compared with the methods *Suspension as Jitter* **(SuspJit)** [9, Page 163] and *Suspension as Blocking* **(SuspBlock)** [9, Page 165]. Moreover, we compared with the Suspension-Oblivious Analysis **(SuspObl)** [9, Page 162] and the Unifying Analysis Framework from Chen, Nelissen, and Huang **(CNH16)** [8] configured with three vectors according to Eq. (27), Lemma 15, and Lemma 16 of their paper. As depicted, our schedulability test performed similar to the state-of-the-art methods.

In Figure 5b, we compare our schedulability test **(EL EDF)** with state-of-the-art methods for EDF. We compared with the method by Liu and Anderson **(LA13)** [26]. Moreover, we



(a) Deadline-Monotonic (DM).



(b) Earliest-Deadline-First (EDF).

Fig. 5: Acceptance ratio of different schedulability tests. Our EDF-Like (EL) schedulability test (EL DM and EL EDF, black curve) performs similar to the state of the art.

show the schedulability test by Günzel, von der Brüggen, and Chen **(GBC20)** [16] and the Suspension-Oblivious Analysis **(SuspObl)** [16, Section III.A]. The method from Dong and Liu [12] is not displayed since it is dominated by **SuspObl**, as shown in [16]. **EL EDF** improves the state of the art.

In Figure 6, the performance of our schedulability test is shown for different configurations for Earliest-Quasi-Deadline-First (EQDF) ($\Pi_i = D_i + \lambda C_i$) and for suspension-aware EDF (SAEDF) ($\Pi_i = D_i + \lambda S_i$). Choosing λ to be the best integer in $[-10, 10]$ improves acceptance ratio compared to the standard EDF with $\lambda = 0$, especially for EQDF.

Figures 7 and 8 show the performance of our schedulability test for arbitrary deadlines. More specifically, we set the deadline to $x = 1.0, 1.1, 1.2, 1.5$ times the minimum inter-arrival time **(Dx)** and applied our schedulability test. We see that both the fixed and the variable analysis window lead to better acceptance ratios in certain scenarios, depending on the size of x and the scheduling algorithm under analysis. From the theoretical discussion in Section IV-D we know that **EL-fix DM 1.0** (**EL-fix EDF 1.0**, respectively) and **EL-var DM 1.0** (**EL-var EDF 1.0**, respectively) coincide. We observe that **EL-var** already benefits from small enlargements of the deadline, whereas **EL-fix** can reach better guarantees for larger deadline extensions in some scenarios. The non-dominance discussed in Section IV-E can be observed in Figure 7 for D1.2 and D1.5.

In Figure 9, we compare the performance of our schedulability test **(EL-fix, EL-var)** with the test by Günzel et al. [17] **(GUC21)**, applying them to arbitrary deadline tasks under DM scheduling. We observe that in the more general case with $[0.8, 1.2]T_i$, **GUC21** outperforms **EL-fix** and **EL-var**. However, as shown in Figure 9b, there are some configurations where **EL-var** performs better than **GUC21**.

Moreover, we examined the **runtime** of our analysis. We created 100 task sets for each utilization in 0% to 100% in steps of 10% and measured the runtime to receive a schedulability decision. To obtain the measurements, we run an implementation with Python3 on a machine with 2x AMD EPYC 7742 running Linux, i.e., in total 256 threads with 2,25GHz and 256GB RAM. Each of the measurements ran on one independent thread. For sets with 200 tasks, our schedulability test took on average 12.87 seconds and at most 17.77 seconds to return the result. The runtime for other relative priority points was comparable.

VI. Conclusion

We study EDF-Like (EL) scheduling algorithms, which include common scheduling strategies like Task-Level Fixed Priority, Earliest Deadline First, and First-In-First-Out. Through an examination of different analysis intervals, we provide two versions of a suspension-aware schedulability test that are valid for all EL scheduling algorithms. We do not assume any restriction for the relation between deadline and period and, to our knowledge, provide the first suspension-aware schedulability test for EL scheduling of arbitrary-deadline tasks. In particular, this is also the first suspension-aware schedulability test for arbitrary-deadline tasks under First-In-First-Out (FIFO), Earliest-Quasi-Deadline-First (EQDF), and Suspension-Aware EDF (SAEDF) scheduling.

Appendix A: Proof for TFP as EL

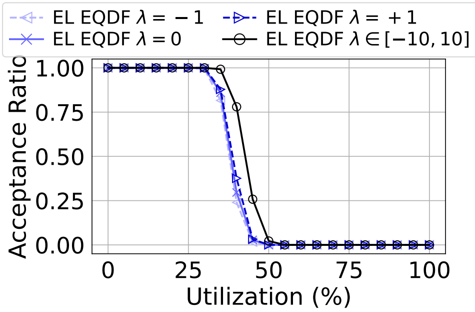
For the sake of completeness, we here provide the proof for the statement that there is an EL algorithm for every TFP algorithm from Section III.

Proposition 16 (TFP as EL.). *Let $K_1, \dots, K_n \in \mathbb{R}_{\geq 0}$ and $\Pi_i := \sum_{j=1}^{i} K_j$ for $i = 1, \dots, n$. If for all $j = 1, \dots, n$ the value K_j is an upper bound on the worst-case response time of τ_j in EL or in TFP, then the schedule of $\mathbb{T}$ under EL and the schedule of $\mathbb{T}$ under TFP coincide.*

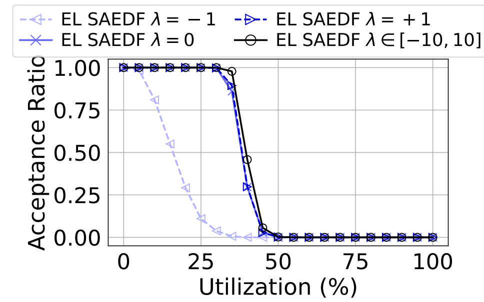
Proof. For an indirect proof we assume that the schedule of $\mathbb{T}$ under EL and TFP does not coincide. Let $\tau_{k,\ell}$ be the job with the highest priority in the EL schedule, such that the schedule of $\tau_{k,\ell}$ does not coincide under EL and TFP. We define the interval

$$I := [r_{k,\ell}, r_{k,\ell} + K_k). \tag{21}$$

Let $\mathbb{J}_{\mathrm{TFP}}$ and $\mathbb{J}_{\mathrm{EL}}$ be the set of jobs with higher priority than $\tau_{k,\ell}$ under TFP and under EL that are executed during I. We

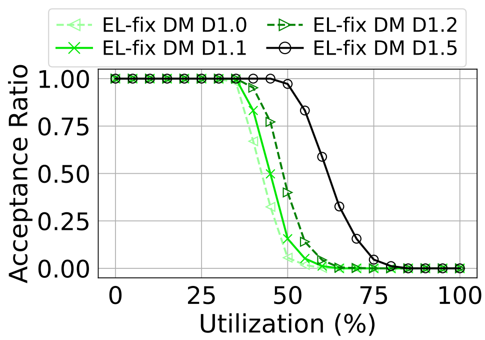


(a) EQDF ($\Pi_i = D_i + \lambda C_i$).

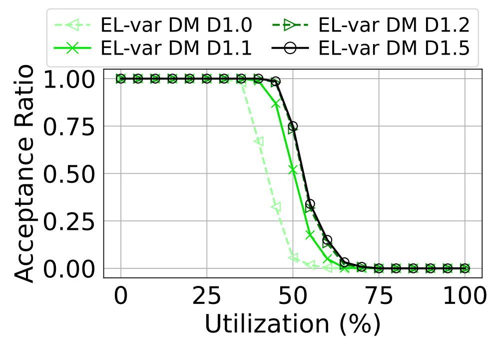


(b) SAEDF ($\Pi_i = D_i + \lambda S_i$).

Fig. 6: Acceptance ratio of variants of EDF using our EDF-Like (EL) schedulability test. Choosing the best $\lambda \in [-10, 10]$ for each task set (black line) improves standard EDF ($\lambda = 0$).

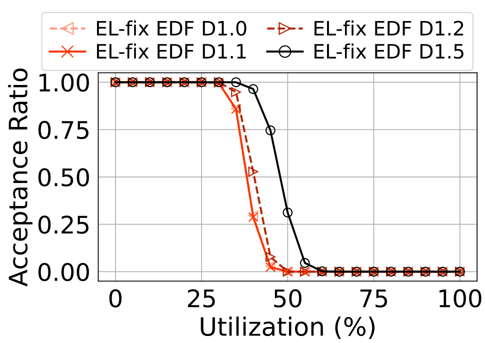


(a) Fixed analysis window.

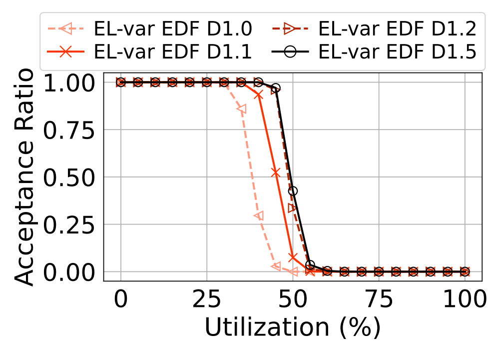


(b) Variable analysis window.

Fig. 7: Arbitrary deadline evaluation for Deadline-Monotonic (DM) scheduling.



(a) Fixed analysis window.



(b) Variable analysis window.

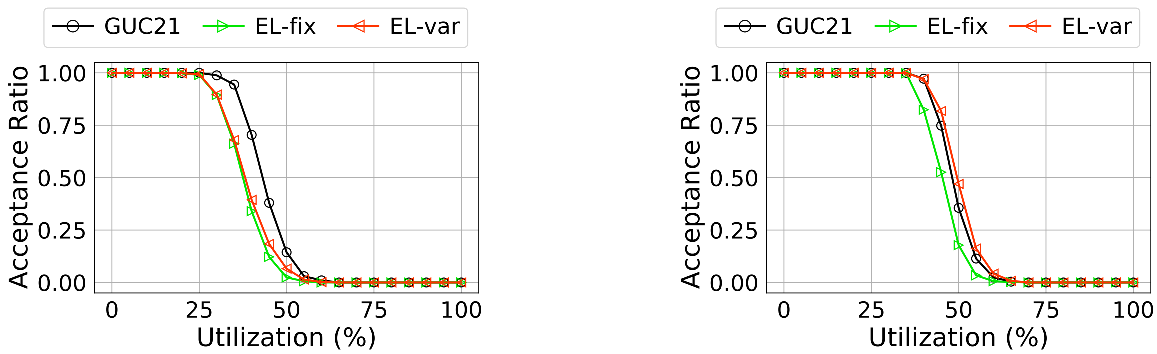
Fig. 8: Arbitrary deadline evaluation for Earliest-Deadline-First (EDF) scheduling.

will reach a contradiction by showing that $\mathbb{J}_{\text{TFP}} = \mathbb{J}_{\text{EL}}$ and that further the schedule of the jobs in $\mathbb{J}_{\text{TFP}} = \mathbb{J}_{\text{TFP}}$ coincides under TFP and under EL scheduling. For that purpose we partition the job sets into three subsets $\mathbb{J}_{\text{TFP}} = \coprod_{i \in \{+,-,0\}} \mathbb{J}^i_{\text{TFP}}$ and $\mathbb{J}_{\text{EL}} = \coprod_{i \in \{+,-,0\}} \mathbb{J}^i_{\text{EL}}$, where each of them denotes the subset of jobs released by tasks of $\mathbb{T}_+ = \{\tau_{k+1}, \dots, \tau_n\}$, $\mathbb{T}_- = \{\tau_1, \dots, \tau_{k-1}\}$ or $\mathbb{T}_0 = \{\tau_k\}$. In the following we show that $\mathbb{J}^i_{\text{TFP}} = \mathbb{J}^i_{\text{EL}}$ for all $i \in \{+,-,0\}$.

Proof of $\mathbb{J}^+_{\text{TFP}} = \mathbb{J}^+_{\text{EL}}$: Because of the task-level priority order under TFP, all jobs of the tasks of $\mathbb{T}_+$ have a lower priority than $\tau_{k,\ell}$, i.e., $\mathbb{J}^+_{\text{TFP}} = \emptyset$. Under EL, we choose any job $\tau_{i,j}$, with $\tau_i \in \mathbb{T}_+$, that has higher priority than $\tau_{k,\ell}$, i.e., $\pi_{i,j} = r_{i,j} + \Pi_i \leq \pi_{k,\ell} = r_{k,\ell} + \Pi_k$ holds. By subtracting Π_k we obtain

$$r_{i,j} + K_i \leq r_{i,j} + (\Pi_i - \Pi_k) \leq r_{k,\ell}. \tag{22}$$

Since $\tau_{i,j}$ has higher priority than $\tau_{k,\ell}$, by assumption the schedule of $\tau_{i,j}$ coincides under EL and TFP, and we have $f_{i,j} \leq r_{i,j} + K_i$. With Equation (22) we conclude $f_{i,j} \leq r_{k,\ell}$. In particular, the job $\tau_{i,j}$ is not executed during I. Since $\tau_{i,j}$



(a) $D_i \in [0.8, 1.2]T_i$.

(b) $D_i \in [1.0, 1.2]T_i$.

Fig. 9: Comparison of arbitrary-deadline schedulability tests under DM scheduling.

was chosen arbitrarily, this means that $\mathbb{J}^+_{\mathrm{EL}} = \emptyset$ as well.

Proof of $\mathbb{J}^-_{\mathrm{TFP}} = \mathbb{J}^-_{\mathrm{EL}}$: Under TFP and under EL, jobs of the tasks in $\mathbb{T}_-$ can only be executed during I if they are released before $r_{k,\ell}+K_k$, i.e., let $\mathbb{J}'^-$ be the set of jobs released before $r_{k,\ell} + K_k$ by tasks of $\mathbb{T}_-$ then $\mathbb{J}^-_{\mathrm{TFP}}, \mathbb{J}^-_{\mathrm{EL}} \subseteq \mathbb{J}'^-$. Under TFP, all jobs in $\mathbb{J}'^-$ have higher priority than $\tau_{k,\ell}$ since they are released by the tasks of $\mathbb{T}_-$. Under EL, we show the same: Let $\tau_{i,j} \in \mathbb{J}'^-$, i.e., $r_{i,j} < r_{k,\ell} + K_k$. It directly follows that

$$\pi_{i,j} = r_{i,j} + \Pi_i < r_{k,\ell} + K_k + \Pi_i \leq r_{k,\ell} + \Pi_k = \pi_{k,\ell}. \quad (23)$$

In particular, $\tau_{i,j}$ has higher priority than $\tau_{k,\ell}$. We have shown that all jobs in $\mathbb{J}'^-$ have higher priority than $\tau_{k,\ell}$ under EL and under TFP scheduling. By assumption the schedule of the jobs in $\mathbb{J}'^-$ coincides under TFP and EL. Therefore the same jobs of $\mathbb{J}'^-$ executed during I under TFP and EL, i.e., $\mathbb{J}^-_{\mathrm{TFP}} = \mathbb{J}^-_{\mathrm{EL}}$.

Proof of $\mathbb{J}^0_{\mathrm{TFP}} = \mathbb{J}^0_{\mathrm{EL}}$: Under TFP and EL, $\mathbb{J}'^0 := \{\tau_{k,1}, \dots, \tau_{k,\ell-1}\}$ are the jobs of τ_k that have higher priority than $\tau_{k,\ell}$, i.e., $\mathbb{J}^0_{\mathrm{TFP}}, \mathbb{J}^0_{\mathrm{EL}} \subseteq \mathbb{J}'^0$. By assumption, the schedule of the jobs in $\mathbb{J}'^0$ coincides. Therefore the same jobs of $\mathbb{J}'^0$ are executed during I, i.e., $\mathbb{J}^0_{\mathrm{TFP}} = \mathbb{J}^0_{\mathrm{EL}}$.

We have shown that $\mathbb{J}_{\mathrm{TFP}} = \mathbb{J}_{\mathrm{EL}}$ by the above discussion. Since the schedule of the jobs $\mathbb{J}_{\mathrm{TFP}} = \mathbb{J}_{\mathrm{EL}}$ coincides during I, $\tau_{k,\ell}$ is preempted/blocked during the same time intervals under TFP and EL scheduling during I. Hence, the schedule of $\tau_{k,\ell}$ during I coincides. Since by assumption K_k is an upper bound on the response time under EL or TFP scheduling, the job $\tau_{k,\ell}$ finishes during I. This proves that the whole schedule of $\tau_{k,\ell}$ coincides. □

Even without knowledge about the worst-case response times, we can use a schedulability test based on EL scheduling for TFP scheduling by setting the relative priority points to $\Pi_i = \sum_{j=1}^{i} D_j$. If the schedulability test assures that all jobs meet their deadline, then D_j is an upper bound on the worst-case response time. In this case, the schedule obtained by EL scheduling coincides with the TFP schedule and is feasible.

Corollary 17. *If the task set $\mathbb{T}$ is schedulable under EL with $\Pi_i := \sum_{j=1}^{i} D_j, i = 1, \dots, n$, then $\mathbb{T}$ is schedulable under TFP as well.*

Proof. If $\mathbb{T}$ is schedulable under EL with the given relative priority points Π_i, then D_j is an upper bound on the worst-case response time of τ_j for all $\tau_j \in \mathbb{T}$ under EL scheduling. In this case, by Proposition 16 the schedule under TFP and EL coincide. Therefore, D_j is also an upper bound on the worst-case response time of τ_j for all $\tau_j \in \mathbb{T}$ under TFP scheduling. Hence, $\mathbb{T}$ is schedulable under TFP as well. □

Acknowledgment

This work has been supported by Deutsche Forschungsgemeinschaft (DFG), as part of Sus-Aware (Project No. 398602212). 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).

References

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

[2] H. Back, H. S. Chwa, and I. Shin. Schedulability analysis and priority assignment for global job-level fixed-priority multiprocessor scheduling. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 297–306. IEEE Computer Society, 2012.

[3] 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.

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

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

[6] J.-J. Chen, T. Hahn, R. Hoeksma, N. Megow, and G. von der Brüggen. Scheduling self-suspending tasks: New and old results. In S. Quinton, editor, *31st Euromicro Conference on Real-Time Systems, ECRTS*, volume 133, pages 16:1–16:23, 2019.

[7] J.-J. Chen and C. Liu. Fixed-relative-deadline scheduling of hard real-time tasks with self-suspensions. In *Real-Time Systems Symposium (RTSS)*, pages 149–160, 2014.

[8] 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)*, 2016.

[9] 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.

[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 *23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2017, Hsinchu, Taiwan, August 16-18, 2017*, pages 1–10. IEEE Computer Society, 2017.

[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] P. Emberson, R. Stafford, and R. I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*, pages 6–11, 2010.

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

[15] M. Günzel and J.-J. Chen. A note on slack enforcement mechanisms for self-suspending tasks. *Real-Time Syst.*, 2021.

[16] M. Günzel, G. von der Brüggen, and J.-J. Chen. Suspension-aware earliest-deadline-first scheduling analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):4205–4216, 2020.

[17] M. Günzel, N. Ueter, and J.-J. Chen. Suspension-aware fixed-priority schedulability test with arbitrary deadlines and arrival curves. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 418–430, 2021.

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

[19] W.-H. Huang and J.-J. Chen. Self-suspension real-time tasks under fixed-relative-deadline fixed-priority scheduling. In *Design, Automation, and Test in Europe (DATE)*, pages 1078–1083, 2016.

[20] 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.

[21] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, May 1986.

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

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

[24] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium'89*, pages 166–171, 1989.

[25] H. Leontyev and J. H. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. In *RTSS*, pages 413–422. IEEE Computer Society, 2007.

[26] 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.

[27] 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.

[28] 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.

[29] J. W. S. Liu. *Real-Time Systems*. Prentice Hall PTR, 1st edition, 2000.

[30] G. Nelissen, J. Fonseca, G. Raravi, and V. Nélis. Timing Analysis of Fixed Priority Self-Suspending Sporadic Tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 80–89, 2015.

[31] G. Nelissen, J. Fonseca, G. Raravi, and V. Nélis. Errata: Timing analysis of fixed priority self-suspending sporadic tasks. Technical Report CISTER-TR-170205, CISTER, ISEP, INESC-TEC, 2017.

[32] B. Peng and N. Fisher. Parameter adaption for generalized multiframe tasks and applications to self-suspending tasks. In *22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA*, pages 49–58. IEEE Computer Society, 2016.

[33] L. Schönberger, W.-H. Huang, G. von der Brüggen, K.-H. Chen, and J.-J. Chen. Schedulability analysis and priority assignment for segmented self-suspending tasks. In *24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA*, pages 157–167. IEEE Computer Society, 2018.

[34] TU Dortmund LS12. EDF-like scheduling schedulability evaluation. https://github.com/tu-dortmund-ls12-rt/EDF-Like, 2022.

[35] G. von der Brüggen, W.-H. Huang, and J.-J. Chen. Hybrid self-suspension models in real-time embedded systems. In *International Conference on Real-Time Computing Systems and Applications (RTCSA)*, 2017.

[36] G. von der Brüggen, W.-H. Huang, J.-J. Chen, and C. Liu. Uniprocessor scheduling strategies for self-suspending task systems. In *International Conference on Real-Time Networks and Systems*, RTNS '16, pages 119–128, 2016.