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={}
}

©2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
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 Leon-tyne 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.

Contributions: 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, like 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 \( T = \{\tau_1, \ldots, \tau_n\} \) of \( n \in \mathbb{N} \) independent self-suspending sporadic real-time tasks in a uniprocessor system. Each task \( \tau_i, i \in \{1, \ldots, n\} \) 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 \( \tau_i \) is denoted by \( r_{i,j} \) and released at time \( r_{i,j} \). We alternate the usage of \( j \) and \( i,j \) to denote the jobs of a task \( \tau_i \) with \( j \in \mathbb{N} \). Each job \( r_{i,j} \) 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 \( \tau_i \) and by \( U := \sum_{i=1}^{n} U_i \) the total utilization of \( T \). A task set \( T \) has constrained-deadlines if \( D_i \leq T_i \) for every \( \tau_i \in T \); 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 \( A \) 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 \( \tau_i \) 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 \( \tau_i \) or \( R_i \leq D_i \) for all \( \tau_i \). A task set is schedulable by a scheduling algorithm \( A \) if for each job sequence released by \( T \), \( A \) 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 \( \pi_{i,j} \). We assume that the jobs of a task \( \tau_i \) finish execution. 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 \( \Pi_i \), denoted as relative priority point, i.e., \( \pi_{i,j} = \pi_{0,j} + \Pi_i \). As a result, smaller \( \Pi_i \) in comparison to the other relative priority points favor the jobs of \( \tau_i \) 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 \( \tau_i \) and \( \tau_{i'} \) in \( T \). Furthermore, let \( \Pi_i \) and \( \Pi_{i'} \) be the relative priority points of \( \tau_i \) and \( \tau_{i'} \). The job \( \tau_{i,j} \) has higher priority than \( \tau_{i',j'} \) if \( \pi_{i,j} < \pi_{i',j'} \). The priority points \( \pi_{i,j} = \pi_{i',j'} + \Pi_i \) and \( \pi_{i',j'} = \pi_{i',j'} + \Pi_{i'} \). 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 \( \tau_i \) must be executed one after another in a FIFO manner. That is, if a job \( \tau_{i,j} \) of task \( \tau_i \) is only eligible for execution (i.e., ready to be executed) after all jobs of \( \tau_i \) 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...

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 \( \pi_{1,1} < \pi_{1,2} < \pi_{2,1} < \pi_{1,3} \).

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).
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. 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] \( (\Pi_i = D_i + \lambda C_i \text{ for some predefined } \lambda \in \mathbb{R}) \)
- Suspension-aware variations of EDF (SAEDF) \( (\Pi_i = D_i + AS_i \text{ for some predefined } \lambda \in \mathbb{R}) \)
- 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 \( \Pi_i \) 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., \( \tau_k \) has higher priority than \( \tau_{k'} \) if and only if \( k < k' \) (with \( \tau_i \) 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 \( \tau_i \) 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 \( T = \{\tau_1, \ldots, \tau_4\} \) of 4 tasks. In the following we demonstrate how to assign priorities such that \( \tau_1 \) and \( \tau_2 \) are on one priority-level, and \( \tau_3 \) and \( \tau_4 \) 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 \( T \) is schedulable under EL scheduling with the given relative priority points, then EL produces the same schedule as the desired scheduling policy: Since \( \Pi_1 = \Pi_j \geq D_i \) for all \( i = 3, 4 \) and \( j = 1, 2 \), a job \( J \) of \( \tau_1 \) or \( \tau_2 \) can only have higher priority than a job \( J' \) of \( \tau_3 \) or \( \tau_4 \) if \( J' \) is already finished when \( J \) is released. \( \tau_1 \) and \( \tau_2 \) are scheduled according to EDF, since their relative priority points are set to the deadline. The tasks \( \tau_3 \) and \( \tau_4 \) 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.

**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 \( T = \{\tau_1, \ldots, \tau_n\} \) and an assignment of relative priority points \( (\Pi_1, \ldots, \Pi_n) \) the test returns True if \( T \) is schedulable by the corresponding EL scheduling algorithm.

Our high-level idea is to bound the worst-case response time of a task \( \tau_k \) 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 \( \tau_k \).

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 \( \tau_k \) 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 \( \sum_{j<\ell} WS_{k,j} \), and for the interference from other tasks, denoted \( B_{k,\ell}' \), 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 \( \tau_k \) 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 \( \tau_k \) at a time instant \( t \), then we call the active job of \( \tau_k \) with the earliest release time the current job.
of \( \tau_k \) at time \( t \). We call the task \( \tau_k \) active at \( t \), if there exists an active job of \( \tau_k \) 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 \((\Pi_1, \ldots, \Pi_n)\) for the task set \( T = \{\tau_1, \ldots, \tau_n\} \). For each task \( \tau_k \) we define the following two **Processor States** (PS):

- \( \text{PS}_b^k \): There is an active job of \( \tau_k \) but the processor is working on a job with higher priority than the current job of task \( \tau_k \), i.e., \( \tau_k \) is **blocked**.
- \( \text{PS}^{ws}_k \): There is an active job of \( \tau_k \) and the processor is working on or suspended by a job of \( \tau_k \).

Whenever there is an active job of \( \tau_k \) and the current job \( \tau_{k,\ell} \) of \( \tau_k \) 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 \( \tau_k \), the processor is in state \( \text{PS}_b^k \) or in state \( \text{PS}^{ws}_k \).

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 \( \tau_k \) 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 \( \text{PS}_b^k \) and \( \text{PS}^{ws}_k \).

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

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

- \( B_{k,\ell}(c, d) \) is the amount of time during \([c, d]\) that the processor is working on jobs of \( \tau_i \) with higher priority than \( \tau_{k,\ell} \).
- \( WS_{k,\ell}(c, d) \) 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 \( \sum_{i \neq k} B_{i,\ell}(c, d) \) is the amount of time during \([c, d]\) that the processor is in state \( \text{PS}_b^k \) and \( WS_{k,\ell}(c, d) \) is the amount of time during \([c, d]\) that the processor is in state \( \text{PS}^{ws}_k \). Moreover, if a previous job \( \tau_{k,j} \), \( j < \ell \) is current during \([c, d]\), then \( \sum_{i \neq k} B_{i,\ell}(c, d) \) is an upper bound on the amount of time during \([c, d]\) that the processor is in state \( \text{PS}_b^k \).

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 \( \tau_k \) is active during (the whole interval) \([c, d]\), then

\[
(C_k + S_k) + \sum_{i \neq k} B_{i,\ell}(c, d) + \sum_{j < \ell} WS_{k,j}(c, d) > (d - c). \tag{1}
\]

**Proof.** By Observation 6, the processor is in state \( \text{PS}_b^k \) or in state \( \text{PS}^{ws}_k \) at all times during \([c, d]\).

Let \( \xi \geq 0 \) be the non-negative integer, such that \( \tau_{k,\ell - \xi}, \ldots, \tau_{k,\ell} \) are the current jobs of \( \tau_k \) 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 \( \sum_{i \neq k} B_{i,\ell}(c_j, d_j) \) and \( WS_{k,j}(c_j, d_j) \) are upper bounds for the amount of time during \([c_j, d_j]\) in state \( \text{PS}_b^k \) and \( \text{PS}^{ws}_k \), respectively, we obtain

\[
d_j - c_j \leq \sum_{i \neq k} B_{i,\ell}(c_j, d_j) + WS_{k,j}(c_j, d_j) \tag{2}
\]

for all \( j \in \{\ell - \xi, \ldots, \ell\} \). Summing up the individual bounds for all \( j \) yields

\[
d - c \leq \sum_{i \neq k} B_{i,\ell}(c, d) + \sum_{j=\ell-\xi}^{\ell} WS_{k,j}(c_j, d_j). \tag{3}
\]

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

\[
d - c \leq \sum_{i \neq k} B_{i,\ell}(c, d) + \sum_{j=\ell} WS_{k,j}(c, d). \tag{4}
\]

Moreover, we have \( WS_{k,\ell}(c, d) < C_k + S_k \) 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 \( \tilde{R}_{k,\ell} \leq D_k \) for \( \tau_{k,\ell} \) by setting \( d = r_{k,\ell} + \tilde{R}_{k,\ell} \). 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 \( \tau_k \) is active during \([c, r_{k,\ell})\). If

\[
\tilde{R}_{k,\ell} := (C_k + S_k) + \sum_{i \neq k} B_{k,\ell}^i(c, d_{k,\ell}) + \sum_{j < \ell} W_{S_{k,j}}(c, d_{k,\ell}) + c - r_{k,\ell},
\]

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

**Proof.** We prove by contradiction and assume that \( \tilde{R}_{k,\ell} \) is not an upper bound on the response time of \( \tau_{k,\ell} \), i.e., the job \( \tau_{k,\ell} \) is not finished at the time \( r_{k,\ell} + \tilde{R}_{k,\ell} \). We set \( d := r_{k,\ell} + \tilde{R}_{k,\ell} \).

Lemma 8 can be applied for the following reasons:

- \( d = r_{k,\ell} + \tilde{R}_{k,\ell} = (C_k + S_k) + \sum_{i \neq k} B_{k,\ell}^i(c, d_{k,\ell}) + \sum_{j < \ell} W_{S_{k,j}}(c, d_{k,\ell}) + c \geq c \) by the definition of \( \tilde{R}_{k,\ell} \) 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 \( \tau_k \) is active during the interval \([c, d)\).

Since Lemma 8 can be applied this means that Equation (1) holds. We have \( B_{k,\ell}^i(c, d) \leq B_{k,\ell}^i(c, d_{k,\ell}) \) for all \( i \neq k \) and \( W_{S_{k,j}}(c, d) \leq W_{S_{k,j}}(c, d_{k,\ell}) \) for all \( j < \ell \) since \( d \leq d_{k,\ell} \). Hence, the left hand side of Equation (1) is upper bounded by \( \tilde{R}_{k,\ell} - c + r_{k,\ell} \). We conclude that \( \tilde{R}_{k,\ell} - c + r_{k,\ell} > d - c = \tilde{R}_{k,\ell} + r_{k,\ell} - c \), 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 \( B_{k,\ell}^i(c, d_{k,\ell}) \) and \( W_{S_{k,j}}(c, d_{k,\ell}) \)? Since computing the values directly has high complexity, we use overapproximation. In Section IV-B we derive upper bounds for \( B_{k,\ell}^i(c, d_{k,\ell}) \) and \( W_{S_{k,j}}(c, d_{k,\ell}) \) 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 \( \tau_k \), and gradually increase the analysis window. In Section IV-E we discuss that these two methods do not dominate each other.

### 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 \( \sum_{j < \ell} W_{S_{k,j}}(c, d_{k,\ell}) \) in Theorem 10 and for \( B_{k,\ell}^i(c, d_{k,\ell}) \) 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 \( \tau_k \) must be executed one after another, i.e., even if the processor idles, a job of \( \tau_k \) cannot start its execution unless all jobs of \( \tau_k \) released prior to it are finished.

We achieve a bound for \( \sum_{j < \ell} W_{S_{k,j}}(c, d_{k,\ell}) \) 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 \( \tau_k \)).** The amount of time during \([c, d_{k,\ell}]\) that the processor is working on or suspended for jobs of \( \tau_k \) with by higher priority than \( \tau_{k,\ell} \) is

\[
\sum_{j < \ell} W_{S_{k,j}}(c, d_{k,\ell}) \leq \left( \left( \frac{d_{k,\ell} - c}{T_k} \right) - 1 \right) (C_k + S_k). \tag{6}
\]

**Proof.** All higher-priority jobs of \( \tau_k \) finish until their deadline. Therefore, the processor can only work on or be suspended by a higher priority job of \( \tau_k \) 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 \( \frac{d_{k,\ell} - c}{T_k} \) jobs of \( \tau_k \) 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 each task \( \tau_i \neq \tau_k \), let \( \bar{R}_i \) be an upper bound on the worst-case response time (WCRT) of jobs of \( \tau_i \) with higher priority than \( \tau_k \). We set \( \bar{R}_i := D_i \) 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 \([\tau_i, d_{k,\ell}]\) that the processor is working on jobs of \( \tau_i \) with higher priority than \( \tau_k \) is

\[
B_{k,\ell}^i(c, d_{k,\ell}) \leq \max \left( \left\{ \frac{G_k' + \bar{R}_i + r_{k,\ell} - c}{T_i} , 0 \right\} C_i \right)
\]

where \( G_k' := \min(D_k - C_i, \Pi_k - \Pi_i) \).

**Proof.** The bound from Equation (7) is achieved by proving the two upper bounds for \( B_{k,\ell}^i \) we get for both cases of \( G_k' \). That is, we prove the following two bounds individually:

\[
B_{k,\ell}^i(c, d_{k,\ell}) \leq \max \left( \left\{ \frac{\Pi_k - \Pi_i + \bar{R}_i + r_{k,\ell} - c}{T_i} , 0 \right\} C_i \right) \tag{8}
\]

\[
B_{k,\ell}^i(c, d_{k,\ell}) \leq \max \left( \left\{ \frac{D_k - C_i + \bar{R}_i + r_{k,\ell} - c}{T_i} , 0 \right\} C_i \right) \tag{9}
\]

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

- Jobs of \( \tau_i \) have higher priority than \( \tau_k \), only if they are released no later than \( r_{k,\ell} + \Pi_k - \Pi_i \).
- Jobs of \( \tau_i \) can be executed after \( c \) only if they are released after \( c - \bar{R}_i \).

Due to these two observations, the number of jobs that contribute to \( B_{k,\ell}^i(c, d_{k,\ell}) \) is upper bounded by the number of releases in \((c - \bar{R}_i, r_{k,\ell} + \Pi_k - \Pi_i)\), which is at most \( \max \left( \left\{ \frac{\Pi_k - \Pi_i + \bar{R}_i + r_{k,\ell} - c}{T_i} , 0 \right\} \right) \). 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 \( \tau_i \) is obtained when the last interfering job of \( \tau_i \) 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 \( B_{k,\ell}^i(c, d_{k,\ell}) \) is therefore upper bounded by the number of releases in \((c - \bar{R}_i, d_{k,\ell} - C_i)\), which is at most \( \max \left( \left\{ \frac{D_k - C_i + \bar{R}_i + r_{k,\ell} - c}{T_i} , 0 \right\} \right) \). 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 \( \tau_i \) during \([c, d_{k,\ell}]\) then \( B_{k,\ell}^i(c, d_{k,\ell}) = 0 \) and the lemma is proven. Otherwise, let \( \tau_{i,j'} \) be the last job of \( \tau_i \) 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

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

and

\[
B_{k,\ell}^i(c, d_{k,\ell}) \leq B_{k,\ell}^i(c, s_{i,j'}) + C^* \tag{11}
\]

We distinguish two cases:

- **Case 1:** \( s_{i,j'} - c < (C_i - C^*) \): The left hand side of Equation (9) is \( B_{k,\ell}^i(c, s_{i,j'}) + C^* \leq (s_{i,j'} - c) + C^* \leq C_i \). Moreover, the right hand side of Equation (9) is at least \( \left\lfloor \frac{D_k - C_i + r_{k,\ell} - c}{T_i} \right\rfloor C_i \geq \left\lfloor \frac{D_k + r_{k,\ell} - c}{T_i} \right\rfloor C_i \geq C_i \) since:
  - \( \bar{R}_i \geq C_i \) by definition of \( \bar{R}_i \).
  - \( \tau_{i,j'} \) is executed during \([c, d_{k,\ell}]\) and therefore \( c < d_{k,\ell} \) holds.

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 \( \tau_i \) for at most \( (C_i - C^*) \) time units, we have \( B_{k,\ell}^i(c, s_{i,j'}) \leq (C_i - C^*) + B_{k,\ell}^i(c + (C_i - C^*), s_{i,j'}) \). Hence, we obtain:

\[
B_{k,\ell}^i(c, d_{k,\ell}) \leq B_{k,\ell}^i(c, s_{i,j'}) + C^* \tag{13}
\]

\[
\leq (C_i - C^*) + B_{k,\ell}^i(c + (C_i - C^*), s_{i,j'}) + C_i^* \tag{14}
\]

\[
= B_{k,\ell}^i(c + (C_i - C^*), s_{i,j'}) + C_i \tag{15}
\]

The number of jobs of \( \tau_i \) that contribute to \( B_{k,\ell}^i(c + (C_i - C^*), s_{i,j'}) \) is at most the number of releases from jobs of \( \tau_i \) during the interval \((c + C_i - C^* - \bar{R}_i, r_{i,j'} - T_i)\), which is

\[
\left\lfloor \frac{r_{i,j'} - T_i - c - C_i + C^* + \bar{R}_i}{T_i} \right\rfloor \leq \left\lfloor \frac{r_{k,\ell} - c - C_i + D_k + \bar{R}_i}{T_i} \right\rfloor - 1.
\]

Therefore, for this case, we conclude that

\[
B_{k,\ell}^i(c, d_{k,\ell}) \leq B_{k,\ell}^i(c + (C_i - C^*), s_{i,j'}) + C_i \leq \left( \left\lfloor \frac{r_{k,\ell} - c - C_i + D_k + \bar{R}_i}{T_i} \right\rfloor - 1 \right) C_i + C_i
\]

\[
= \left\lfloor \frac{r_{k,\ell} - c - C_i + D_k + \bar{R}_i}{T_i} \right\rfloor C_i.
\]

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

**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 \( B_{k,\ell}^i(c, d_{k,\ell}) \) and \( WS_{k,j}(c, d_{k,\ell}) \) provided in the previous section to obtain the following schedulability test.
Algorithm 1 Schedulability test with fixed analysis window.

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

1: Order $\tau_1, \ldots, \tau_n$ s.t. $D_1 \geq \cdots \geq D_n$.
2: Set $R_k := D_k$ for all $k$.
3: for $i = 1, 2, \ldots, \text{depth}$ do
4:  $\text{solved} := \text{True}$
5:  for $k = 1, 2, \ldots, n$ do
6:      $\text{cand} := [1]$; $\text{step} := \eta \cdot D_k$ \quad $\triangleright$ Preparation.
7:      for $b = 0, \text{step}, 2 \cdot \text{step}, \ldots < D_k$ do \quad $\triangleright$ Compute.
8:      $\text{cand} := \text{append}(\text{cand})$ \quad $\triangleright$ Compare candidates.
9:      $R_k := \min(\text{cand})$ \quad $\triangleright$ Check condition.
10:     if $R_k > D_k$ then break
11:     $\text{solved} := \text{False}; \tilde{R}_k := D_k$; break
12: return solved

Theorem 12 (Sufficient Schedulability Test). Let $T = \{\tau_1, \ldots, \tau_n\}$ 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

\[
\tilde{R}_k(b_k) \leq D_k,
\]

where $\tilde{R}_k(b_k) := \max_{i \neq k} \left( \left\lfloor \frac{G_i^k + \tilde{R}_k b_k}{T_i} \right\rfloor, 0 \right) C_i + \frac{D_k - b_k}{T_k} (C_k + S_k) + b_k$ and $G_i^k = \min(D_k - C_i, \Pi_k - \Pi_k)$, then the task set is schedulable by EL scheduling with the given relative priority points and the worst-case response time of $\tau_k$ is upper bounded by $\tilde{R}_k(b_k)$.

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 $T$ and denote by $\text{Seq}$ the sequence of all jobs in the schedule ordered by their priority. Via induction, we prove that the first $\xi$ jobs in $\text{Seq}$ have the required response time upper bound, for all $\xi \in \mathbb{N}_0$. Consequently, $\tilde{R}_k \leq D_k$ and the task set is schedulable.

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

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

\[
(C_k + S_k) + \sum_{i \neq k} \max \left( \left\lfloor \frac{G_i^k + \tilde{R}_k + r_{k,\xi} - c}{T_i} \right\rfloor, 0 \right) C_i + \left( \frac{d_k,\xi - c}{T_k} - 1 \right) \cdot (C_k + S_k) + c - r_{k,\xi}.
\]

When choosing $c := b_k + r_{k,\xi}$ we obtain that $\tilde{R}_k$ is upper bounded by $\tilde{R}_k$. Due to Equation (16), we know $\tilde{R}_k \leq \tilde{R}_k \leq D_k$, i.e., the job meets its deadline. We use Theorem 9 to conclude that $\tilde{R}_k$ is an upper bound on the response time of $\tau_{k,\xi+1}$ and therefore $\tilde{R}_k$ is an upper bound on the response time of $\tau_{k,\xi+1}$ 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 $\tilde{R}_k$ for every task $\tau_k$ 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 $\tilde{R}_k$ for every task $\tau_k$ correctly and efficiently.

To determine the values of $b_k$, we take a user-specified parameter $\eta$ and discretize the search space into $\frac{1}{\eta}$ values with a step size $\eta \cdot D_k$. To determine $\tilde{R}_k$, we go through the task set several times and compute upper bounds for the values of $\tilde{R}_k$ 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 $\tau$, 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 $\tilde{R}_k = D_k$ 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 $\text{step} = \eta \cdot D_k$, i.e., a certain share of $D_k$ like 1 percent, we compute $\tilde{R}_k(b_k)$ for $b_k = 0, 1 \cdot \text{step}, 2 \cdot \text{step}, \ldots$ until $b \geq D_k$ is reached. We then take the minimal value of all these candidates and define it as the new $\tilde{R}_k$. The time complexity of Algorithm 1 is $O\left(\frac{\text{depth} \cdot n^2}{\eta}\right)$.

Please note that the computed values of $\tilde{R}_k$ are in fact only upper bounds of $\tilde{R}_k$ from Theorem 12. A reduction of $\tilde{R}_k$, $i \neq k$ in subsequent iterations reduces the actual value of $\tilde{R}_k$ as well, since $\tilde{R}_k$ is monotonically increasing with respect to $\tilde{R}_k$, 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 $\tau_{k,\xi-a}$ can only interfere with $\tau_{k,\xi}$ if $\tau_k$
is active during \([r_{k,ℓ}−a,r_{k,ℓ})\). 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 \([c,d_{k,ℓ})\) with \(c \in [r_{k,ℓ}−a,d_{k,ℓ})\).

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 \(τ_{k,ℓ}\) is the \((a+1)\)-th job in an active interval of \(τ_k\), if the following two conditions hold.

1. \(τ_{k,ℓ}\) is active during \([r_{k,ℓ}−a,f_{k,ℓ})\).
2. At time \(r_{k,ℓ}−a\) there is no active job which is released before \(r_{k,ℓ}−a\).

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

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

\[
\min (a+1, \left(1+\frac{d_{k,ℓ}−c}{T_k}\right)(C_k + S_k) + \sum_{i ≠ k} \max \left(\frac{G^*_k + \bar{R}_i + r_{k,ℓ}−c}{T_i}, 0\right) C_i + c − τ_{k,ℓ}) \tag{18}
\]

is at most \(D_k\), with \(G^*_k := \max(D_k − C_k, Π_k − Π_k)\), then (18) is an upper bound on the response time of \(τ_{k,ℓ}\).

**Proof.** For the proof, we apply the analysis backbone from Theorem 9. Since \(τ_{k,ℓ}\) is the \((a+1)\)-th job in an active interval, \(τ_k\) is active during \([r_{k,ℓ}−a,d_{k,ℓ})\). Hence, the restriction on \(c\) in the formulation of Theorem 9 is fulfilled if \(c\) is chosen from the interval \([r_{k,ℓ}−a,d_{k,ℓ})\). Moreover, since \(τ_{k,ℓ}\) is chosen \((a+1)\)-th job in an active interval, the jobs \(τ_{1,ℓ},\ldots,τ_{k−1,ℓ}−a\) are finished by time \(r_{k,ℓ}−a\). We obtain

\[
\sum_{j < ℓ−a} WS_{k,j}(c,d_{k,ℓ}) \leq \sum_{j < ℓ−a} WS_{k,j}(r_{k,ℓ}−a,d_{k,ℓ}) = 0.
\]

Hence, \(\sum_{j < ℓ−a} WS_{k,j}(c,d_{k,ℓ}) = \sum_{j = ℓ−a}^{ℓ−1} WS_{k,j}(c,d_{k,ℓ}) \leq \sum_{j = ℓ−a}^{ℓ−1} (C_k + S_k) = a \cdot (C_k + S_k).\) We combine this upper bound on \(\sum_{j < ℓ} WS_{k,j}(c,d_{k,ℓ})\) with the upper bounds from Lemma 10 and Lemma 11, and obtain that \(\bar{R}_k,ℓ\) 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 \(\overline{R}_{k,ℓ} ≤ D_k\). The analysis backbone from Theorem 9 states that \(\overline{R}_{k,ℓ}\) is an upper bound on the response time of \(r_{k,ℓ}\) and therefore, also Equation (18) is an upper bound on the response time.

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 \([r_{k,ℓ}−aT_k,r_{k,ℓ}+D_k)\).

**Theorem 15 (Sufficient Schedulability Test).** Let \(T = \{τ_1,\ldots,τ_n\}\) be an arbitrary-deadline task set with relative priority points \((Π_1,\ldots,Π_n)\). We define the function \(\bar{R}^a_k : [0,aT_k + D_k) \rightarrow \mathbb{R}_{≥0}\) by the assignment

\[
x \mapsto \min (a+1, \left[\frac{D_k − x + aT_k}{T_k}\right](C_k + S_k) + \sum_{i ≠ k} \max \left(\frac{G^*_k + \bar{R}_i − x + aT_k}{T_i}, 0\right) C_i + x − aT_k).
\]

(19)

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

\[
\bar{R}^a_k(b^a_k) ≤ D_k,
\]

and furthermore \(\bar{R}^{\bar{a}_k}_k(b^{\bar{a}_k}) ≤ T_k\),

(20)

then the task set is schedulable by EL scheduling with the given relative priority points and \(\bar{R}_k := \max_{a = 0,\ldots,\bar{a}_k} \bar{R}^a_k(b^a_k)\) 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 \(ξ \in \mathbb{N}_0\) jobs in \(Seq\):

(i) \(\bar{R}_k\) is a response time upper bound for all jobs of \(τ_k\).
(ii) \(T_k\) is a response time upper bound for all \(\bar{a}_k\)-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:** \(ξ \rightarrow ξ + 1\). The first \(ξ\) jobs in \(Seq\) have the required response time upper bounds (i) and (ii) by induction. We denote by \(τ_{k,ξ}\) the \((ξ + 1)\)-th job of \(Seq\). Let \(a\) be the lowest value in \(\mathbb{N}_0\), such that \(τ_{k,ξ}\) is the \((a+1)\)-th job in an active interval of \(τ_k\).

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

\[\text{Preprint Version. Citation Info:}\]
Algorithm 2 Schedulability test with var. analysis window.

Input: \( T = \{\tau_1, \ldots, \tau_n\}, (\Pi_1, \ldots, \Pi_n), \eta, \text{max}_a, \text{depth}\)

Output: True: schedulable; False: no decision

1: Order \( \tau_1, \ldots, \tau_n \) s.t. \( D_1 \geq \cdots \geq D_n \).
2: Set \( R_k := D_k \) for all \( k \).
3: for \( i = 1, 2, \ldots, \text{depth} \) do
4:     solved := True
5:     for \( k = 1, 2, \ldots, n \) do
6:         for \( a = 0, 1, \ldots, \text{max}_a \) do
7:             cand := \{\} |
8:             step := \eta \cdot D_k
9:             \( \text{cand.append}(\text{cand}) \) from Equation (19)
10:            if \( \text{cand} \leq T_k \) then
11:                \( \text{Check cond. 1.} \)
12:           break
13:       if \( \text{cand} > D_k \) or \( a = \text{max}_a \) then
14:           solved := False; \( R_k := D_k \);
15:           break
16:     return solved

Algorithm 2 depicts an implementation of the schedulability test in pseudocode. As the value of \( \text{cand} \) is in the interval \( a \cdot T_k \), the program may never return a result. To make the schedulability test deterministic, we introduce an additional parameter \( \text{max}_a \) which aborts the loop when no result is obtained for \( a = 0, 1, \ldots, \text{max}_a \).

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 \( x = b_k \cdot T_k \) in Theorem 15, then the result is lower bounded by \( \hat{R}_k(b_k) \) 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 \( \hat{R}_k \) 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 \( \text{max}_a \) this effect is even intensified: The analysis with variable analysis window has to find \( \hat{R}_k \leq T_k \) even for some \( a \leq \text{max}_a \). Third, the discretization using \( \eta \) 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 \( \eta = 0.01, \text{depth} = 5 \), and \( \text{max}_a = 10 \). 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 \([1, 100][\text{ms}]\). 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 \( S_j \) 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 (SuspObli) [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
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) (EI = Dl + λC) and for suspension-aware EDF (SAEDF) (EI = Dl + λSi). Choosing λ to be the best integer in [−10, 10] improves acceptance ratio compared to the standard EDF with λ = 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], 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 K₁, . . . , Kn ∈ R≥0 and \( \Pi_i := \sum_{j=1}^{i} K_j \) for i = 1, . . . , n. If for all j = 1, . . . , n the value \( K_j \) is an upper bound on the worst-case response time of \( \tau_j \) in EL or in TFP, then the schedule of T under EL and the schedule of T under TFP coincide.

**Proof.** For an indirect proof we assume that the schedule of 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

\[
I := [r_{k,\ell}, r_{k,\ell} + K_k].
\]  

Let \( J_{TFP} \) and \( J_{EL} \) be the set of jobs with higher priority than \( \tau_{k,\ell} \) under TFP and under EL that are executed during \( I \). We
In particular, the job $\tau_{i,j}$ is not executed during $I$. Since $\tau_{i,j}$ has higher priority than $\pi_{k,\ell}$, i.e., $\tau_{i,j} \in T_+$, that has higher priority than $\pi_{k,\ell}$, i.e., $\tau_{i,j} \in T_+$, we obtain

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

Since $\tau_{i,j}$ has higher priority than $\pi_{k,\ell}$, 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 $\tau_{i,j}$

---

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

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

---

### 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.

#### (a) Fixed analysis window.

#### (b) Variable analysis window.
was chosen arbitrarily, this means that \( J^\cup_{\text{EL}} = 0 \) as well.

**Proof of \( J^\cup_{\text{TFP}} = J^\cup_{\text{EL}} \):** 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 \( J^- \) be the set of jobs released before \( r_{k,\ell} + K_k \) by tasks of \( \mathbb{T}^- \) then \( J^\cup_{\text{TFP}} \subseteq J^- \). Under TFP, all jobs in \( J^- \) have higher priority than \( \tau_{k,\ell} \) since they are released by the tasks of \( \mathbb{T}^- \). Under EL, we show the same:

Let \( \tau_{i,j} \in J^- \), i.e., \( r_{i,j} < r_{k,\ell} + K_k \). It directly follows that

\[
\pi_{i,j} = r_{i,j} + \Pi_i < r_{k,\ell} + K_k + \Pi_i \leq r_{k,\ell} + \Pi_k = \pi_{k,\ell}.
\]

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

**Proof of \( J^0_{\text{TFP}} = J^0_{\text{EL}} \):** Under TFP and EL, \( J^0 := \{ \tau_{k,1}, \ldots, \tau_{k,L_{\ell,\ell}} \} \) are the jobs of \( \tau_{k,\ell} \) that have higher priority than \( \tau_{k,\ell} \), i.e., \( J^0_{\text{TFP}} \subseteq J^0 \). By assumption, the schedule of the jobs in \( J^0 \) coincide. Therefore the same jobs of \( J^0 \) are executed during \( I \), i.e., \( J^0_{\text{TFP}} = J^0_{\text{EL}} \).

We have shown that \( J^\cup_{\text{TFP}} = J^\cup_{\text{EL}} \) by the above discussion. Since the schedule of the jobs \( J^\cup_{\text{TFP}} = J^\cup_{\text{EL}} \) 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 \( \Pi_i := \sum_{j=1}^{i} D_j, i = 1, \ldots, n \), then \( \mathbb{T} \) is schedulable under TFP as well.

**Proof.** If \( \mathbb{T} \) is schedulable under EL with the given relative priority points \( \Pi_i \), then \( D_j \) is an upper bound on the worst-case response time of \( \tau_j \) for all \( \tau_j \in \mathbb{T} \). 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 \( \tau_j \) for all \( \tau_j \in \mathbb{T} \). 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**


