Suspension-Aware Fixed-Priority Schedulability Test with Arbitrary Deadlines And Arrival Curves

Mario Günzel, Niklas Ueter, Jian-Jia Chen
TU Dortmund, Department of Computer Science, Dortmund, Germany

Citation: 10.1109/RTSS52674.2021.00045

BibTeX:

```
@inproceedings{guenzel21rtss_arrcurve,
  author={Mario Günzel and Niklas Ueter and Jian-Jia Chen},
  booktitle={42nd IEEE Real-Time Systems Symposium (RTSS)},
  title={Suspension-Aware Fixed-Priority Schedulability Test with Arbitrary Deadlines
  And Arrival Curves},
  year={2021},
  volume={},
  number={},
  pages={},
  doi={}
}
```

Suspension-Aware Fixed-Priority Schedulability Test with Arbitrary Deadlines And Arrival Curves

Mario Günzel
TU Dortmund University
mario.guenzel@tu-dortmund.de

Niklas Ueter
TU Dortmund University
niklas.ueter@tu-dortmund.de

Jian-Jia Chen
TU Dortmund University
jian-jia.chen@cs.tu-dortmund.de

***Abstract*—In real-time scheduling theory, self-suspension describes the behavior that a job can suspend itself from the ready state and thus be exempted from the scheduling for the suspension duration. This behavior makes it non-trivial to resort to established concepts such as the busy-interval analysis to self-suspending task sets which is required to analyze the worst-case response time of tasks with backlog, e.g., arbitrary-deadline task sets. In this paper, we present a novel suspension-aware busy-interval analysis for dynamic self-suspension tasks where the inter-arrival time of subsequent jobs can be bounded by an arrival curve. Based on the general analysis, we provide worst-case response time analyses and hence sufficient schedulability tests for fixed-priority preemptive uniprocessor scheduling algorithms for arrival-curve constrained and sporadic self-suspension task systems with arbitrary deadlines. Moreover, we provide evaluations based on synthetically generated task sets that show that our method indeed exploits the optimism that is introduced when enlarging the relative deadline of tasks. We demonstrate that our approach improves the state of the art by considering arrival curves that are obtained from tasks with release jitter.**

I. Introduction

Real-time systems are characterized by the requirement to satisfy specified temporal constraints for mission-critical computations in the systems. These temporal constraints must be formally verified beforehand under consideration of a model of the generated workloads and a model of a given scheduling algorithm. In current real-time systems these temporal constraints are given in terms of task deadlines, which have to be met by each task instance (job). A commonly used class of scheduling algorithms in real-time operating systems are fixed-priority scheduling algorithms in which each task is assigned a static priority that each job inherits. Despite the fact that fixed-priority scheduling algorithms are not optimal they are widely supported by real-time operating systems for uniprocessor systems or partitioned scheduling algorithms in multiprocessor systems. This is due to an intuitive parameterization in terms of priorities that influence the response time of a task. Moreover, the fact that the scheduling operations can be implemented in $\mathcal{O}(1)$ complexity with low overhead enable fixed-priority schedulers to be used on resource-constrained platforms.

Since the work on rate-monotonic scheduling of periodic task sets under real-time constraints by Liu and Layland [33], many results have been published in the literature in pursuance to understand and formalize the behavior of fixed-priority uniprocessor scheduling systems and the analysis of worst-case response times. Based on the critical instant theorem, the time-demand analysis as proposed by Joseph and Pandya [23] and Lehoczky et al. [30] provides exact analyses of the worst-case response times of tasks, which have never more than one unfinished job at any time, scheduled upon uniprocessor systems using fixed-priority scheduling. In the case of arbitrary-deadline task systems in which tasks may have more than one unfinished job at each time (backlog), the time-demand analysis was extended to the busy-interval analysis by Lehoczky [29]. The underlying assumption in all of the above analyses is that no job can yield its ready state during any time between a job release and that jobs completion. In many real-world applications however, this model is insufficient to describe application demands. For instance, offloading parts of the computation to hardware accelerators [12], [35] or the partitioned scheduling of parallel DAG task with sub job partitioning [14] are cases in which a job yields its ready state while waiting for offloaded computations or the completion of preceding sub jobs before resuming to the ready state again. In the literature, such behavior is referred to as *self-suspension* as a job may *suspend* itself from the ready state and resume at a later point in time.

Attempts to extend existing techniques [10], [26] and insights such as the critical instant theorem to self-suspending task systems have been shown to be non-trivial. Under the assumption that a task may self-suspend, many "*[...] key insights underpinning the analysis of non-self-suspending tasks no longer hold*." as detailed in the self-suspension review paper by Chen et al. [9]. In consequence, several research results have been found to be flawed, e.g., the extension of the critical instant theorem in [26] was later proven unsound by Nelissen et al. [37] and the extension of the schedulability test for the earliest-deadline-first (EDF) scheduling algorithm in [10] was proven incorrect by Günzel and Chen [16]. The irregular interference behavior of self-suspending task systems makes it mandatory to construct worst-case response time analyses from first principles.

To date, dynamic and segmented self-suspension are the predominately studied models in the literature as explained by Chen et al. [9]. The segmented self-suspension model [3], [6], [7], [17], [21], [25], [37]–[39] and the dynamic self-

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).

suspension model [1], [8], [10], [16], [18], [22], [32] differ in the allowed suspension pattern. In the segmented self-suspension model, computation and suspension segments are interleaved in a predefined manner and each segment's duration is bounded. In contrast, the dynamic self-suspension model is a generalization of the segmented model in the sense that any interleaved sequence of computation and suspension is admissible as long as the cumultive duration of computation and suspension is bounded. In addition to these two models, von der Brüggen et al. [45] proposed a hybrid self-suspension model, which increases the flexibility of the segmented model and improves accuracy of the dynamic model.

The current state-of-the-art of schedulability analysis for dynamic self-suspending task systems under task-level fixed-priority (FP) scheduling is given by Chen et al. [8]. Other analyses are presented in [1], [22], [24] and [34, Page 162]. However, the results in [1], [24] have been disproved [9] due to the fact that the classical critical instant theorem does not hold for self-suspending tasks. With regards to suspension-aware schedulability tests for task-level dynamic-priority scheduling algorithms, results have been limited to the earliest-deadline-first (EDF) algorithm, in which the priority of each job is given by its absolute deadline. Devi [10] provided a schedulability test for EDF without a proof which has recently been disproved by Günzel and Chen [16] presenting a concrete counter example. Liu and Anderson [31] and Dong and Liu [11] studied global EDF on multiprocessor systems and provided schedulability tests. More recently a dedicated suspension-aware EDF uniprocessor analysis was presented by Günzel et al. [18], which improves the schedulability for uniprocessor EDF significantly compared to the analyses in [11], [31]. With respect to the approximation quality of fixed-priority (FP), earliest-deadline first (EDF), least-laxity-first (LLF), and earliest-deadline-zero-laxity (EDZL) scheduling algorithms have been shown to not have a constant speedup factor when suspension can not be sped up [5] implying that self-suspension may degrade schedulability.

In contrast to prior research, our work focuses on the worst-case response time analysis of dynamic self-suspending task systems with arbitrary deadlines, i.e., the deadline can be larger than the minimum arrival-time, which is to the best of our knowledge the first analysis of this kind. Since the segmented self-suspension model is a specialization of the dynamic self-suspension model, all presented results in this paper can also be applied to the segmented self-suspension model. The state-of-the-art analysis of the studied problem is due to Chen et al. [8], in which a unifying response time analysis was developed. Their analysis allows to either include the suspension times of higher-priority tasks explicitly in the analysis or include more workload modeled by a jitter term induced by self-suspension.

However, their analysis has been limited to constrained-deadline task systems, i.e., the deadline must be no more than the minimum inter-arrival time. For arbitrary-deadline tasks, we identify a *suspension-aware busy-interval* and prove that in the analysis interval at most one self-suspending job (carry-in) of each higher-priority task must be considered. To the best of our knowledge it is an open problem if a busy-interval equivalent concept can be established for self-suspending arbitrary-deadline task systems. Moreover, it is unclear how many jobs must be considered in the analysis even if a busy-interval concept would exist. Current analyses for self-suspending task systems solely focus on systems with periodic or sporadic inter-arrival times of two subsequent jobs. We here use the more general *arrival curve* model known from Real-Time Calculus, Network Calculus [28], [43], and Compositional Performance Analysis (CPA) [19], [20] to model task activations. In the arrival curve model, the maximum (and minimum) number of events for any interval of a given length is provided in order to describe the online arrival behavior of tasks. Since an arrival curve is a more general description than periodic or sporadic inter-arrival times, there also do not exist any dedicated results for self-suspending tasks with arrival curve descriptions in the literature, except using the worst-case execution time to model the additional carry-in workload due to self-suspension.

Contributions: In this paper, we study the worst-case response time analysis problem and devise a schedulability test to validate the worst-case timing behavior for a set of self-suspending arbitrary-deadline tasks given fixed-priority preemptive uniprocessor scheduling algorithms. In conclusion, we make the following contributions:

- We extend the concept of busy-interval by Lehoczky [29] to *suspension-aware busy-interval*. Specifically, the analysis extends the window of interest in a way such that at most *one self-suspending job* of each higher-priority task has to be considered in the analysis.
- We provide the first worst-case response time analysis for self-suspending real-time tasks described by arrival curves in Section IV. The interference from higher-priority self-suspending tasks can be arbitrarily modelled with one of two types of carry-in terms, in which one has self-suspending behavior and one does not.
- Section V handles the special case when the worst-case (upper) arrival curve of a task is periodic, i.e., the classical sporadic real-time tasks, each defined by the minimum inter-arrival time of two consecutive jobs.
- In Section VII we compare our approach with the state of the art for arbitrary-deadline tasks where the arrival curve is modeled for tasks with release jitter. Moreover, we demonstrate that our analysis exploits the optimism that is obtained when increasing the tasks' relative deadlines.

II. System Model

We consider a task set $\mathbb{T}$ consisting of n recurrent tasks $\tau_1, \ldots, \tau_n$. Each task τ_i recurrently releases jobs $\tau_{i,j},\ j \in \mathbb{N}$. A job $\tau_{i,j}$ is released at time $r_{i,j}$ and has to be executed for a certain amount of time $c_{i,j}$ until its deadline $d_{i,j}$ while it suspends itself several times for a total amount of $s_{i,j}$ time units. We call the time from release $r_{i,j}$ to finish $f_{i,j}$ of a job $\tau_{i,j}$ its response time.

Definition 1 (Sporadic Task)**.** A sporadic task τ_i is characterized by a tuple (C_i, S_i, D_i, T_i). More specifically, $C_i > 0$ is the worst-case execution time (WCET) of τ_i, $S_i \geq 0$ is the maximal suspension time, $D_i > 0$ is the relative deadline, and $T_i \geq 0$ is the minimum inter-arrival time, i.e., $r_{i,j+1} \geq r_{i,j} + T_i$, $c_{i,j} \leq C_i$, $s_{i,j} \leq S_i$ and $d_{i,j} = r_{i,j} + D_i$ for all $j \in \mathbb{N}$. There are no constraints between deadline and minimum inter-arrival time of a task, i.e., $D_i > T_i$ is allowed. The worst-case response time (WCRT) R_i of a task τ_i is defined by the supremum over the response times of all its jobs $\tau_{i,j},\ j \in \mathbb{N}$.

As a more general model, we further model the job arrivals of sporadic tasks using arrival curves. For each task τ_i, an upper arrival curve $\alpha_i^u : \mathbb{R} \to \mathbb{R} \geq 0$ is given, which provides by $\alpha_i^u(\Delta)$ an upper bound on the job releases inside an interval of length Δ. More specifically, we have $\alpha_i^u(\Delta) \geq \sup_t(\text{number of job releases during } [t, t+\Delta))$ for all $\Delta \geq 0$ and $\alpha_i^u(\Delta) = 0$ for all $\Delta < 0$.

For a sporadic task whose minimum inter-arrival time is T_i, the corresponding upper arrival curve is $\alpha_i^u(\Delta) = \left\lceil \frac{\Delta}{T_i} \right\rceil$ for any $\Delta \geq 0$. In case T_i is 0, it is possible to release two (or more) jobs at the same time. Modeling such a task with a tuple (C_i, S_i, D_i, T_i) results in an infinite burst and is therefore infeasible. Therefore, for sporadic real-time tasks, we assume to have $T_i > 0$ in Section V. However, when the upper arrival curve is considered in this paper, we assume $T_i \geq 0$ in Section IV.

We consider that the given upper arrival curve function α_i^u is sub-additive [28], i.e., $\alpha_i^u(\Delta_1 + \Delta_2) \leq \alpha_i^u(\Delta_1) + \alpha_i^u(\Delta_2)$, $\forall \Delta_1, \Delta_2 \in \mathbb{R}$. If the given curve is not sub-additive, then it can be tightened by applying the sub-additive closure to achieve the sub-additivity property, i.e., by defining a new upper arrival curve $\tilde{\alpha}_i^u(\Delta) := \min(\alpha_i^u(\Delta), \min_{t \in [0,\Delta]}(\alpha_i^u(t) + \alpha_i^u(\Delta - t)))$. Due to the sub-additivity property, we have that for all $\Delta \in \mathbb{R}$:

1) $\alpha_i^u(\Delta) \leq \alpha_i^u(\Delta + \delta)$ for all $\delta \geq 0$ (α_i^u monotonically increasing); otherwise, the existence of $\delta \geq 0$ with $\alpha_i^u(\Delta) > \alpha_i^u(\Delta + \delta)$ contradicts the sub-additivity property $\alpha_i^u(\Delta + \delta) \leq \alpha_i^u(\Delta) + \alpha_i^u(\delta) \leq \alpha_i^u(\Delta)$.
2) $\alpha_i^u(\Delta) \leq \alpha_i^u(\Delta - T_i) + 1$, which is due to the sub-additivity property $\alpha_i^u(\Delta) \leq \alpha_i^u(\Delta - T_i) + \alpha_i^u(T_i) = \alpha_i^u(\Delta - T_i) + 1$ when $T_i > 0$ and due to the fact that $\alpha_i^u(\Delta) \leq \alpha_i^u(\Delta) + 1$ when T_i is 0.

On the basis of the worst-case response time R_i of a given task τ_i, the maximal amount of concurrently pending workload that is generated by τ_i is given by $C_i^* := \min(\alpha_i^u(R_i) \cdot C_i, R_i)$. In particular, this means that at each time there is no more than $C_i^* \in [C_i, R_i]$ amount of pending workload of jobs of τ_i in a ready state (ready queue). The reason for this is that the maximal number of concurrent jobs of τ_i which are released but not yet finished is upper bounded by $\alpha_i^u(R_i)$. Each of them can have pending workload of up to C_i time units. This shows $C_i^* \leq \alpha_i^u(R_i) \cdot C_i$. Moreover, if the concurrently pending workload at some time instant t would be higher than R_i, then the most recently released job would finish no earlier than at $t + R_i$, i.e., R_i would not be an upper bound of the worst-case response time of τ_i, which is a contradiction. Therefore, $C_i^* \leq R_i$ as well.

III. Problem Definition

In this work, we consider a task set $\mathbb{T}$ consisting of tasks that are adhere to the characterization depicted in Section II. We assume that each task is assigned a unique priority that is given beforehand. Moreover, we focus on a work-conserving preemptive fixed-priority (FP) scheduling on a uniprocessor system, meaning that any released job that is not finished and does not suspend itself, i.e., which is *ready* to be executed on the processor, is subject to the scheduling decision. In other words, as long as there exists at least one *ready* job, the scheduler keeps the processor busy. In the remainder of this paper, we assume that the task set is ordered according to the priority level, i.e., a task τ_i has higher priority than a task τ_j iff $i < j$.

This work answers the question whether jobs released by the task set $\mathbb{T}$ are schedulable according to fixed-priority preemptive scheduling, i.e., all jobs meet their deadline. More specifically, we provide a sufficient schedulability test that returns *True* when we can guarantee that the task set is schedulable. If the test returns *False* the task set might or might not be schedulable.

Although arrival curves are considered, we note that this paper *does not* deal with the modular/compositional performance analysis, the main feature of Real-Time Calculus (RTC) [28], [43] and Compositional Performance Analysis (CPA) [19], [20]. We focus only on the response time analysis (and the schedulability test). In RTC, Greedy Processing Component (GPC) is one of the fundamental components, which processes jobs (input events) in a greedy manner following the first-in-first-out (FIFO) policy. As long as there are available resources, a GPC does not allow any suspension. The analysis of GPC has been studied in [4], [15], [27], [40]–[42], in which the improvements have focused on efficiency and precision.

For CPA, the analysis is based on holistic worst-case schedules, which rely on the busy-interval concept (called longest scheduling horizon in CPA [20]). When analyzing self-suspending tasks, one possibility is to apply the jitter-based analysis by considering that at most $\alpha_i^u(\Delta + R_i)$ jobs of a higher-priority task τ_i can interfere with the task τ_k under analysis, as shown in Equation (1) in [36]. Under fixed-priority preemptive uniprocessor scheduling, it has been shown that converting suspension into computation for task τ_k is analytically sound in [9] when analyzing the worst case of τ_k. Therefore, this treatment results in a jitter-based CPA analysis. This unfortunately does not consider the characteristics of suspension behavior of the higher-priority tasks.

Our analysis presented in this paper is the first suspension-aware analysis for arbitrary-deadline sporadic real-time task systems and arrival curves under fixed-priority uniprocessor scheduling. Extending the analysis to be integrated into CPA or RTC is not part of this paper.

IV. Schedulability Test for Arrival Curves

In this section we derive a sufficient schedulability test. The proof structure is inspired by the constrained-deadline analysis in [8]. However, the characterizations of arbitrary-deadline tasks are completely different from constrained-deadline tasks, and none of their lemmas can be directly applied without proper modifications. We start by considering a fixed-priority uniprocessor preemptive schedule Ψ of the task set $\mathbb{T}$. Iteratively, we consider the task $\tau_k,\ k = 1, \ldots, n$ and provide an upper bound on the worst-case response time for τ_k under the assumption that the upper bound for $\tau_1, \ldots, \tau_{k-1}$ has been derived beforehand.

Let τ_k be the task under analysis. In the following, we consider some job $\tau_{k,\ell}$ and bound its response time. We partition the higher priority tasks $\tau_1, \ldots, \tau_{k-1}$ into two sets denoted by $\mathbb{T}_0$ and $\mathbb{T}_1$. Our analysis assumes that this partition is given and provides a valid sufficient schedulability test. Depending on the partition the analysis approach in Section IV-A is different: For each task we either cut or extend the analysis window. To find a suitable partition, one can examine all of them. We explain how to find suitable partitions in Section VI.

The proof is divided into four steps:

Step 1: Reducing the schedule Ψ by removing jobs that do not contribute to the response time of $\tau_{k,\ell}$.

Step 2: Analyzing the reduced schedule Ψ^1 and proving useful properties for later analysis.

Step 3: Providing a response time upper bound for $\tau_{k,\ell}$.

Step 4: Deriving the schedulability test.

Our analysis is based on the *suspension-aware busy-interval* of τ_k defined as follows:

Definition 2 (Suspension-aware busy-interval)**.** The half-opened interval $[v, w)$ is a *suspension-aware busy-interval* of τ_k if there is pending workload of task τ_k at all times during that interval $[v, w)$.

Let $a \in \mathbb{N}$. The job $\tau_{k,\ell}$ is the *a-th job in a suspension-aware busy-interval* of τ_k if $[r_{k,\ell-(a-1)}, f_{k,\ell})$ is a suspension-aware busy-interval of τ_k and all jobs of τ_k released before $r_{k,\ell-(a-1)}$ finish at latest at $r_{k,\ell-(a-1)}$. □

Example 3. In Figure 1 we present an example schedule of three tasks $\mathbb{T} = \{\tau_1, \tau_2, \tau_3\}$ with $\mathbb{T}_0 = \{\tau_1\}$ and $\mathbb{T}_1 = \{\tau_2\}$ to give the reader guidance through the proof. The job $\tau_{3,2}$ is under analysis. It is the second job in a suspension-aware busy-interval of τ_3.

A. Step 1: Reducing the schedule Ψ

At first we remove all tasks with lower priority than τ_k from the system. This does not affect the schedule of $\tau_1, \ldots, \tau_k$ as the lower priority tasks are anyway preempted when there is pending workload of $\tau_1, \ldots, \tau_k$. Afterwards, we remove further jobs from the schedule. In this step, the removal of jobs in the schedule has to be guaranteed not to affect the response time of $\tau_{k,\ell}$ in the schedule.

Let $a \in \mathbb{N}$ such that $\tau_{k,\ell}$ is the a-th job in a *suspension-aware busy-interval* of τ_k, as defined in Definition 2. We

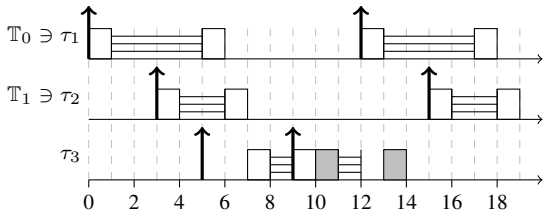


Figure 1: Schedule Ψ of 3 tasks from Example 3. The execution of job $\tau_{3,2}$ is marked gray.

set $t_k := r_{k,\ell-(a-1)}$ as the beginning of the suspension-aware busy-interval and remove the jobs $\tau_{k,1}, \ldots, \tau_{k,\ell-a}$ from the schedule. The removal of these jobs does not affect the schedule of the higher-priority tasks, as the higher priority tasks would preempt job execution of τ_k anyway. Moreover, since τ_k has the lowest-priority in the schedule Ψ (after the treatment in the first paragraph) there are no lower priority tasks to be affected. The jobs released at or after t_k (including $\tau_{k,\ell}$) are not affected as well, since all removed jobs are finished until time t_k.

We define Ψ^k to be the resulting reduced schedule. In the following, we describe how to derive Ψ^i and t_i from Ψ^{i+1} and t_{i+1} iteratively, for $i = k-1, k-2, \ldots, 1$. The main procedure is to *extend the analysis window* if $\tau_i \in \mathbb{T}_1$ and to *cut the overlapping job* of τ_i if $\tau_i \in \mathbb{T}_0$. More specifically, we distinguish four different cases as depicted in Figure 2. Please note that the four cases cover all possible scenarios.

Procedure from Ψ^{i+1} to Ψ^i:

Case X0: All jobs of τ_i are released at or after t_{i+1}. In this case, we define $\Psi^i := \Psi^{i+1}$ and $t_i := t_{i+1}$. □

The other three cases (**X1**, **X2**, and **X3**) involve scenarios in which there are jobs of τ_i released before t_{i+1}. We denote by c_i^* the *residual workload* at time t_{i+1} of τ_i, i.e., the amount of remaining time that the processor needs to work on pending jobs of τ_i at time t_{i+1}. Moreover, let τ_{i,ℓ_i} be the first job of τ_i which finishes after t_{i+1}.

Case X1: $\tau_i \in \mathbb{T}_1$ and $r_{i,\ell_i} \leq t_{i+1}$. In this case we set $t_i := \max(f_{i,\ell_i-1}, r_{i,\ell_i})$ to be maximum of release of the first job that finishes after t_{i+1} and the finish of the previous job. We remove all jobs of τ_i before τ_{i,ℓ_i}. □

Case X2: $\tau_i \in \mathbb{T}_1, r_{i,\ell_i} > t_{i+1}$. In this case we set $t_i := t_{i+1}$ and remove all jobs of τ_i released before τ_{i,ℓ_i}. Note that afterwards there is no job execution of or job release of τ_i before t_{i+1}. □

Case X3: $\tau_i \in \mathbb{T}_0$. In this case we define $t_i := t_{i+1}$. All jobs released after t_i remain unmodified in the schedule. All jobs released before t_i are replaced by *one* artificial job with execution time c_i^* and release t_i with same priority and the same execution and suspension behavior as the other jobs had after t_{i+1}. In particular, the execution and suspension pattern of τ_i after t_i remains unchanged and there is no job release before t_i. □

Please note that the transformation from Ψ^{i+1} to Ψ^i does not affect the response time of $\tau_{k,\ell}$ due to the following rea-

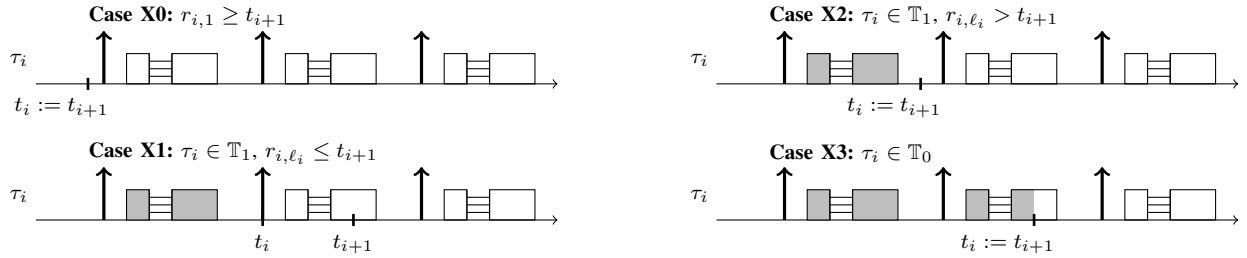


Figure 2: Procedure from Ψ^{i+1} to Ψ^i as in Step 1. Gray areas are removed from the schedule.

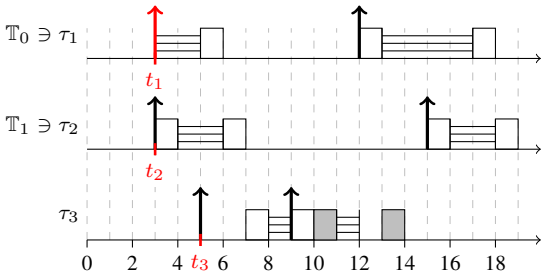


Figure 3: Schedule (Ψ^1) from Example 3 after Step 1.

soning: In the procedure from Ψ^{i+1} to Ψ^i, only the schedule of τ_i before t_i is modified. Before t_i there are no jobs of lower priority tasks released (all of them are already removed in Ψ^{i+1}). Therefore, modifying the schedule of τ_i has no impact on the lower priority tasks. As a result, the job $\tau_{k,\ell}$ is not affected by the transformation from Ψ^{i+1} to Ψ^i. We conclude the following.

Lemma 4. *The response time of $\tau_{k,\ell}$ in Ψ coincides with the response time of $\tau_{k,\ell}$ in Ψ^1.*

Proof: In this subsection we discussed that neither the transformation from Ψ to Ψ^k nor the transformation from Ψ^{i+1} to Ψ^i for any $i \in \{1, \ldots, k-1\}$ affects the response time of $\tau_{k,\ell}$. Since the transformation from Ψ to Ψ^1 is just a composition of the above transformations, this does not affect the response time of $\tau_{k,\ell}$ as well. ■

The procedure of this subsection is illustrated by the following example.

Example 5. In Figure 3 we present the schedule Ψ^1 for the original schedule from Figure 1. We set t_3 to the release of the first job in the suspension-aware busy-interval of the job under analysis $\tau_{3,2}$ at time 5. When going from Ψ^3 to Ψ^2 we set t_2 according to Case X1 to time 3. Finally, we set $t_1 := t_2$ and cut the job as presented for Case X3. We obtain Ψ^1.

B. Step 2: Analyzing Ψ^1

To deduce a worst-case response time of $\tau_{k,\ell}$, in this step we analyze the amount of time that the processor executes and idles in Ψ^1. Therefore, we introduce the following notation.

Definition 6. For any interval $[c, d)$, we define $\text{idle}(c, d)$ and $\text{exec}(c, d)$ to be the amount of time that the processor is idle and executing some job during $[c, d)$ in Ψ^1. Moreover, we define $\text{exec}_i(c, d)$ to be the amount of time a job of task τ_i is executed during $[c, d)$ in Ψ^1. In particular we have $\text{exec}(c, d) = \sum_{i=1}^{n} \text{exec}_i(c, d)$. □

The backbone for this step of the analysis is the fact that for any interval $[t_1, t)$ with $t_k \leq t \leq f_{k,\ell}$, we have

$$\text{idle}(t_1, t) + \text{exec}(t_1, t) = t - t_1. \tag{1}$$

In the following we derive upper bounds for $\text{idle}(t_1, t)$ and $\text{exec}(t_1, t)$, and conclude the response time upper bound. We utilize $\text{idle}(t_1, t) = \sum_{i=1}^{k-1} \text{idle}(t_i, t_{i+1}) + \text{idle}(t_k, t)$ and $\text{exec}(t_1, t) = \sum_{i=1}^{k} \text{exec}_i(t_1, t)$. Moreover, we have $\text{exec}_i(t_1, t) = \text{exec}_i(t_i, t)$ for all i since there is no job of τ_i being executed before t_i in Ψ^1.

Lemma 8 shows that for each segment $[t_i, t_{i+1})$ the idle time is upper bounded by suspension time in Ψ^1. In the proof we utilize that the original schedule Ψ provides a work-conserving property.

Definition 7 (Work-Conserving)**.** A schedule is called *work-conserving*, if it fulfills the following property for any task τ_i: Whenever there is pending workload of task τ_i, then either a job of a higher priority task is executed or a job of τ_i is executed or suspends itself. □

Lemma 8. *Consider the schedule Ψ^1.*

1) *Let $i \in \{1, \ldots, k-1\}$. At any time instant during $[t_i, t_{i+1})$ a job of τ_i is executed or suspends itself, or a job from a higher priority task is executed.*
2) *At any time instant during $[t_k, f_{k,\ell})$ a job of τ_k is executed or suspends itself, or a job from a higher priority tasks is executed.*

Proof: The schedule Ψ is work-conserving. By the construction of t_i in Section IV-A, at any time instant during $[t_i, t_{i+1})$ there is pending work of τ_i. More specifically, either $t_i = t_{i+1}$ or there is a job of τ_i which is released no later than t_i and finishes after t_{i+1}. Due to the work preserving property, at all time instants during $[t_i, t_{i+1})$ in the schedule Ψ a job of τ_i is executed or suspends itself, or a job of a higher priority task is executed. When we construct Ψ^1 from Ψ, during $[t_i, t_{i+1})$ we only remove execution and suspension from jobs of lower priority tasks. Hence, 1) holds.

During $[t_k, f_{k,\ell})$ in Ψ, there is a suspension-aware busy-interval of task τ_k, i.e., there is pending work of τ_k at all time

instants during $[t_k, f_{k,\ell})$. Similar as above, due to the work preserving property, at all time instants during $[t_k, f_{k,\ell})$ in Ψ a job of τ_k is executed or suspends itself, or a job of a higher priority task is executed. However, when Ψ^1 is constructed from Ψ, during $[t_k, f_{k,\ell})$ only execution and suspension from jobs of lower priority tasks is removed. Hence, 2) holds. ∎

This lemma implies that each time the processor idles during $[t_i, t_{i+1})$ a job of task τ_i suspends itself and each time the processor idles during $[t_k, f_{k,\ell})$ a job of task τ_k suspends itself. Let x_i be 1 if $\tau_i \in \mathbb{T}_1$ and 0 if $\tau_i \in \mathbb{T}_0$. We define by $\text{susp}_i(t_i, t_{i+1})$ the amount of time that jobs of τ_i suspend during $[t_i, t_{i+1})$ in Ψ^1.

Lemma 9. *For any $i = 1, \dots, k-1$ we have $idle(t_i, t_{i+1}) \leq x_i \cdot susp_i(t_i, t_{i+1}) \leq x_i \cdot S_i$.*

Proof: If t_i and t_{i+1} coincide, then by definition the equations $\text{idle}(t_i, t_{i+1}) = 0$ and $\text{susp}_i(t_i, t_{i+1}) = 0$ hold. Moreover, $x_i, S_i \geq 0$ which concludes this case.

If t_i and t_{i+1} do not coincide, then $t_i < t_{i+1}$. This can only be achieved if Case X1 is applied when going from Ψ^{i+1} to Ψ^i, i.e., $x_i = 1$. It remains to show that $\text{idle}(t_i, t_{i+1}) \leq \text{susp}_i(t_i, t_{i+1}) \leq S_i$. Since $t_i = \max(f_{i,\ell_i-1}, r_{i,\ell_i})$, all jobs of τ_i prior to τ_{i,ℓ_i} finish at or before t_i. Moreover, τ_{i,ℓ_i} finishes after t_{i+1}. Hence, τ_{i,ℓ_i} is the only job of τ_i that suspends itself during $[t_i, t_{i+1})$, i.e., $\text{susp}_i(t_i, t_{i+1}) \leq S_i$. Due to Lemma 8, during $[t_i, t_{i+1})$ a job of τ_i suspends itself whenever the processor idles. Hence, $\text{idle}(t_i, t_{i+1}) \leq \text{susp}_i(t_i, t_{i+1})$. ∎

Utilizing the second part of Lemma 8, we provide a similar statement about the idle time during $[t_k, t)$ for any $t \in [t_k, f_{k,\ell}]$. However, during this interval there are a many jobs of τ_k that may suspend themselves.

Lemma 10. *For any $t \in [t_k, f_{k,\ell}]$ we have $idle(t_k, t) \leq a \cdot S_k$.*

Proof: Due to Lemma 8, whenever the processor idles during the interval $[t_k, t)$ then there is some job of τ_k that suspends itself. By the definition of t_k, the interval $[t_k, f_{k,\ell})$ is a suspension-aware busy-interval of τ_k with a jobs. Hence, there can be at most a jobs of τ_k that suspend themselves during $[t_k, t) \subset [t_k, f_{k,\ell})$ and $\text{idle}(t_k, t) \leq a \cdot S_k$. ∎

After an estimation of the idle time of the processor during $[t_1, t)$, we focus on the execution time. For each task $\tau_i \neq \tau_k$, we do this by setting Δ to $t - t_i$ and providing an upper bound for $\text{exec}_i(t_i, t_i + \Delta)$. In Lemma 11 we present a general bound which is applicable to all tasks. Afterwards, in Lemma 14 we consider the case that $\tau_i \in \mathbb{T}_1$, in Lemma 15 we consider the case that $\tau_i \in \mathbb{T}_0$, and in Lemma 16 we consider the case $\tau_i = \tau_k$.

Lemma 11. *Let τ_i in $\mathbb{T}$. For any $\Delta \geq 0$, we have*

$$exec_i(t_i, t_i + \Delta) \leq \alpha_i^u(\Delta + R_i) \cdot C_i \tag{2}$$

where R_i is an upper bound on the worst-case response time of the task τ_i.

Proof: To be executed during the interval $[t_i, t_i + \Delta)$ a job of τ_i must be released before $t_i + \Delta$. Moreover, it must not be finished until t_i, which is only possible if it is released after $t_i - R_i$. We conclude that only jobs that are released during the interval $(t_i - R_i, t_i + \Delta)$ may be executed during $[t_i, t_i + \Delta)$. The number of those jobs is upper bounded by $\alpha_i^u(\Delta + R_i)$. Each of them can be executed for at most C_i time units. ∎

For a precise proof of Lemma 14, 15 and 16, we first introduce the notation of interference I_i and derive useful properties in Lemma 13:

Definition 12 (Interference)**.** For all $t \in \mathbb{R}$ and $\Delta \geq 0$, we define by $I_i(t, t+\Delta)$ the *interference* during the interval $[t, t+\Delta)$ from task τ_i, which is the amount of execution time during $[t, t+\Delta)$ from jobs of τ_i which are released during $[t, t+\Delta)$.

Moreover, we define $I_i^u(\Delta)$ to be the *maximum interference* from task τ_i during an interval of length Δ, i.e.,

$$I_i^u(\Delta) = \begin{cases} \sup_t I_i(t, t+\Delta) & \Delta \geq 0 \\ 0 & \Delta < 0 \end{cases}. \tag{3}$$

Lemma 13. *For the maximum interference function I_i^u the following properties hold for all $\Delta \geq 0$:*

1) $I_i^u(\Delta) \leq I_i^u(\Delta - \delta) + \delta$ for all $\delta \geq 0$
2) $I_i^u(\Delta) \leq \alpha_i^u(\Delta) \cdot C_i$

Proof: Let $\Delta, \delta \geq 0$ be fixed. During an interval of length δ, there can be at most δ amount of workload being executed. Therefore, for all t we have $I_i(t, t+\Delta) - I_i(t, t+\Delta-\delta) \leq \delta$. By using the supremum, we obtain the first part of the lemma.

The maximum number of job releases during an interval of length Δ is upper bounded by $\alpha_i^u(\Delta)$. Each of these jobs can be executed for at most C_i time units. ∎

In the following three lemmas, we provide the the upper bound for $\text{exec}_i(t_i, t_i + \Delta)$ if $\tau_i \in \mathbb{T}_1$, $\tau_i \in \mathbb{T}_0$ or $\tau_i = \tau_k$.

Lemma 14. *Let τ_i in $\mathbb{T}_1$. For any $\Delta \geq 0$, we have*

$$exec_i(t_i, t_i + \Delta) \leq \alpha_i^u(\Delta + \max(R_i - T_i, 0)) \cdot C_i \tag{4}$$

where R_i is an upper bound on the worst-case response time of the task τ_i.

Proof: Since $\tau_i \in \mathbb{T}_1$, t_i can be derived by **Case X0**, **Case X1** or **Case X2** during the procedure from Ψ^{i+1} to Ψ^i.

If t_i is derived by **Case X0**, then all jobs of τ_i are released at or after t_i. Therefore, $\text{exec}_i(t_i, t_i + \Delta) = I_i(t_i, t_i + \Delta) \leq I_i^u(\Delta) \leq \alpha_i^u(\Delta) \cdot C_i$. Due to the monotonicity of the arrival curve, we obtain the result from Equation (4).

If t_i is derived by **Case X1**, then all jobs of τ_i prior to τ_{i,ℓ_i} are finished at time t_i and are therefore not executed after t_i. If $t_i = r_{i,\ell_i}$, then all jobs that execute after t_i are released at or after t_i, similar to **Case X0**. More specifically, we have $\text{exec}_i(t_i, t_i + \Delta) = I_i(t_i, t_i + \Delta) \leq I_i^u(\Delta) \leq \alpha_i^u(\Delta) \cdot C_i \leq \alpha_i^u(\Delta + \max(R_i - T_i, 0)) \cdot C_i$. If $t_i = f_{i,\ell_i-1}$, then τ_{i,ℓ_i} is released no earlier than $t_i - R_i + T_i$. We obtain $\text{exec}_i(t_i, t_i + \Delta) \leq I_i(t_i - R_i + T_i, t_i + \Delta) \leq I_i^u(\Delta + R_i - T_i) \leq \alpha_i^u(\Delta + R_i - T_i) \cdot C_i$. Since the arrival curve α_i^u is monotonically increasing, replacing $R_i - T_i$ by $\max(R_i - T_i, 0)$ yields the result.

If t_i is derived by **Case X2**, then all jobs of τ_i which are executed after t_i are released at or after t_i. Analogously to

Case X0, we obtain $\text{exec}_i(t_i, t_i + \Delta) = I_i(t_i, t_i + \Delta) \leq I_i^u(\Delta) \leq \alpha_i^u(\Delta) \cdot C_i \leq \alpha_i^u(\Delta + \max(R_i - T_i, 0)) \cdot C_i$ as in Equation (4). ∎

We note that the bound from Lemma 14 is tighter than the bound from Lemma 11, since $\alpha_i^u(\Delta + \max(R_i - T_i, 0)) \leq \alpha_i^u(\Delta + \max(R_i, 0)) \leq \alpha_i^u(\Delta + R_i)$.

Lemma 15. *Let τ_i in $\mathbb{T}_0$. For any $\Delta \geq 0$, we have*

$$\text{exec}_i(t_i, t_i + \Delta) \leq \alpha_i^u(\Delta - T_i + R_i - C_i^*) \cdot C_i + C_i^* \quad (5)$$

where R_i is an upper bound on the worst-case response time of τ_i and $C_i^* = \min(\alpha_i^u(R_i) \cdot C_i, R_i)$ is an upper bound on the maximum current workload as defined in Section II.

Proof: Since $\tau_i \in \mathbb{T}_0$, t_i can be derived by **Case X0** or **Case X3** during the procedure from Ψ^{i+1} to Ψ^i.

If t_i is derived by **Case X0**, then analogously to the proof of Lemma 14 we obtain $\text{exec}_i(t_i, t_i + \Delta) \leq \alpha_i^u(\Delta) \cdot C_i$. According to Section II, for the arrival curve we have $\alpha_i^u(\Delta) \leq \alpha_i^u(\Delta - T_i) + 1$. Due to the monotonicity of α_i^u and since $R_i - C_i^* \geq 0$, $\alpha_i^u(\Delta - T_i)$ is less than or equal to $\alpha_i^u(\Delta - T_i + R_i - C_i^*)$. by using $C_i \leq C_i^*$ we obtain the result from Equation (5).

If t_i is derived by **Case X3**, then $c_i^* \leq C_i^*$ denotes the residual workload at time t_i. Let $\tau_{i,\ell_i}, \ldots, \tau_{i,\ell_i+p}$ be the jobs that contribute to c_i^*, then τ_{i,ℓ_i+p} finishes no earlier than $f_{i,\ell_i+p} \geq t_i + c_i^*$. Since $f_{i,\ell_i+p} \leq r_{i,\ell_i+p} + R_i \leq r_{i,\ell_i+p+1} - T_i + R_i$, we obtain that τ_{i,ℓ_i+p+1} and all following jobs are released no earlier than $r_{i,\ell_i+p+1} \geq t_i + c_i^* + T_i - R_i$. This yields $\text{exec}_i(t_i, t_i + \Delta) \leq c_i^* + I_i^u(\Delta - c_i^* - T_i + R_i)$. Lemma 13 with δ set to $C_i^* - c_i^*$ yields the inequality $\text{exec}_i(t_i, t_i + \Delta) \leq C_i^* + I_i^u(\Delta - C_i^* - T_i + R_i)$ which is at most $C_i^* + \alpha_i^u(\Delta - C_i^* - T_i + R_i) \cdot C_i$. ∎

In general, the bounds from Lemma 11 and Lemma 15 do not dominate each other. Hence, when $\tau_i \in \mathbb{T}_0$ both bounds have to be considered.

Lemma 16. *For $t \in [t_k, f_{k,\ell}]$ we have $\text{exec}_k(t_k, t) \leq a \cdot C_k$.*

Proof: The interval $[t_k, f_{k,\ell})$ is a suspension aware busy-interval of τ_k with a jobs. Hence, there are only a jobs of τ_k that can be executed by the processor during $[t_k, f_{k,\ell})$. As a result, $\text{exec}_k(t_k, t) < \text{exec}_k(t_k, f_{k,\ell}) \leq a \cdot C_k$. ∎

For τ_k we have an upper bound for the idle time and execution time formulated in Lemma 10 and Lemma 16 when $t \in [t_k, f_{k,\ell}]$. If $t \neq f_{k,\ell}$, i.e., $\tau_{k,\ell}$ is not already finished at time t, we know that there is remaining execution or suspension time from $\tau_{k,\ell}$. In this case we provide the following lemma.

Lemma 17. *For all $t \in [t_k, f_{k,\ell})$ we have $\text{exec}_k(t_k, t) + \text{idle}(t_k, t) < a \cdot (C_k + S_k)$.*

Proof: We proof this lemma by contradiction and assume that there exists one $t \in [t_k, f_{k,\ell})$ such that $\text{exec}_k(t_k, t) + \text{idle}(t_k, t) \geq a \cdot (C_k + S_k)$. Since $\text{exec}_k(t_k, t) \leq a \cdot C_k$ and $\text{idle}(t_k, t) \leq a \cdot S_k$ by Lemmas 16 and 10, both take their highest value, i.e., $\text{exec}_k(t_k, t) = a \cdot C_k$ and $\text{idle}(t_k, t) = a \cdot S_k$. We conclude that all a jobs of τ_k in the suspension-aware busy-interval finished their complete execution time. Moreover, $\text{idle}(t_k, t) \leq \text{susp}_k(t_k, t)$ by Lemma 8. In particular, all a jobs suspend themselves for S_k time units each. As a result, all a jobs of τ_k are finished and $t \geq f_{k,\ell}$. This contradicts the assumption. ∎

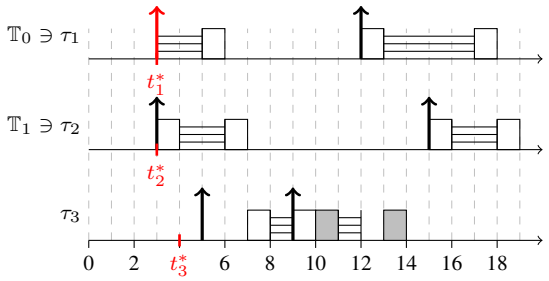


Figure 4: Schedule from Example 3 after Step 3.

C. Step 3: Provide Response Time Upper Bound

In this step we provide a response time upper bound for the job $\tau_{k,\ell}$. For this purpose, we safely enlarge the intervals for which we estimate the execution time from $[t_i, t)$ to $[t_i^*, t)$. We do this to ensure the property $t_{i+1}^* - t_i^* = \text{idle}(t_i, t_{i+1}) \leq S_i$, which is then utilized to bound the left boundary of the analysis intervals $[t_i^*, t)$. Subsequently, we compose the upper bounds from Step 2 to obtain a response time bound.

Definition 18. Iteratively, we define

$$t_1^* := t_1 \quad (6)$$

$$t_i^* := t_{i-1}^* + x_{i-1} \cdot \text{idle}(t_{i-1}, t_i) \quad (7)$$

for all $i = 2, \ldots, k$, where $\text{idle}(t_{i-1}, t_i)$ is the amount of idle time during $[t_{i-1}, t_i)$ in the schedule Ψ^1.

Example 19. In Figure 4 we present the choice of t_i^* that is obtained from the schedule Ψ^1 in Figure 3. We start by setting $t_1^* := t_1 = 3$. Since there is no idle time between t_1 and t_2 in Ψ^1, we define $t_1^* := t_2^*$. Between t_2 and t_3 the processor idles for one time unit, namely during the interval $[4, 5)$. We set $t_3^* := t_2^* + 1 = 4$.

We start the analysis by proving a simple property which is later used to describe the boundaries of the analysis intervals.

Lemma 20. *For all $i = 1, \ldots, k$ we have $t_i^* \leq t_i$.*

Proof: This follows from Definition 18 using the fact that $\text{idle}(t_{i-1}, t_i) \leq (t_i - t_{i-1})$. ∎

With the following definition the execution time bounds from Step 2 can be summarized by $\text{exec}_i(t_i, t_i + \Delta) \leq A_i^1(\Delta)$ whenever $\tau_i \in \mathbb{T}_1$ and $\leq A_i^0(\Delta)$ whenever $\tau_i \in \mathbb{T}_0$.

Definition 21 (A_i^1 and A_i^0)**.** We define the A_i^1 and A_i^0 by

$$A_i^1(\Delta) := \alpha_i^u(\Delta + \max(R_i - T_i, 0)) \cdot C_i \quad (8)$$

$$A_i^0(\Delta) := \min \left(\begin{array}{l} \alpha_i^u(\Delta + R_i) \cdot C_i, \\ \alpha_i^u(\Delta - T_i + R_i - C_i^*) \cdot C_i + C_i^* \end{array} \right) \quad (9)$$

for all $\Delta \in \mathbb{R}$, where $C_i^* = \min(\alpha_i^u(R_i) \cdot C_i, R_i)$ as defined in Section II.

For the response time upper bound we formulate a property that only holds when $t < f_{k,\ell}$, in the following lemma. Whenever the property does not hold, we assure that the finishing time must be exceeded. In particular, this allows to indicate response time upper bounds.

Lemma 22. *For all $t \in [t_k^*, f_{k,\ell})$ the inequality*

$$a \cdot (C_k + S_k) + \sum_{i=1}^{k-1} \left(\begin{array}{l} x_i \cdot A_i^1(t - t_i^*) \\ +(1 - x_i) A_i^0(t - t_i^*) \end{array} \right) > t - t_k^* \quad (10)$$

holds.

Proof: Equation (1) states that $\text{idle}(t_1, t) + \text{exec}(t_1, t) = t - t_1$. We know that $\text{idle}(t_1, t) = \text{idle}(t_1, t_k) + \text{idle}(t_k, t)$ and $t_1 + \text{idle}(t_1, t_k) = t_k^*$. Hence, subtracting $\text{idle}(t_1, t_k)$ in Equation (1) yields

$$\text{idle}(t_k, t) + \text{exec}(t_1, t) = t - t_k^*. \quad (11)$$

For the execution part, $\text{exec}(t_1, t) = \sum_{i=1}^{k-1} \text{exec}_i(t_i, t) + \text{exec}(t_k, t)$ holds. By Lemma 14 and Lemma 15,

$$\text{exec}_i(t_i, t) \leq x_i \cdot A_i^1(t - t_i) + (1 - x_i) \cdot A_i^0(t - t_i) \quad (12)$$

for all $i < k$ since $x_i = 1$ iff $\tau_i \in \mathbb{T}_1$ and $x_i = 0$ iff $\tau_i \in \mathbb{T}_0$.

Since the arrival curve is monotonically increasing, A_i^1 and A_i^0 are monotonically increasing as well. Hence, (12) is upper bounded by $x_i \cdot A_i^1(t - t_i^*) + (1 - x_i) \cdot A_i^0(t - t_i^*)$ since $t_i^* \leq t_i$ by Lemma 20. Using this together with the bound from Lemma 17 yields the result from Equation (10). ∎

In the following lemma, we make the property in Equation (10) independent from t_i^* by introducing $Q_i^{\vec{x}}$.

Lemma 23. *For $i = 1, \ldots, k-1$ we define $Q_i^{\vec{x}} := \sum_{j=i}^{k-1} x_j S_j$. The inequality*

$$a \cdot (C_k + S_k) + \sum_{i=1}^{k-1} \left(\begin{array}{l} x_i \cdot A_i^1(\theta + Q_i^{\vec{x}}) \\ +(1 - x_i) A_i^0(\theta + Q_i^{\vec{x}}) \end{array} \right) > \theta \quad (13)$$

holds for all $\theta \in [0, f_k - t_k^*)$.

Proof: We obtain Equation (13) be replacing the variable t in Lemma 22 by $\theta + t_k^*$, i.e., $\theta = t - t_k^*$. Moreover, we have $(t_k^* - t_i^*) \leq Q_i^{\vec{x}}$ since $(t_k^* - t_i^*) = \sum_{j=i}^{k-1} x_j \cdot \text{idle}(t_j, t_{j+1}) \leq \sum_{j=i}^{k-1} x_j \cdot S_j$ because of Lemma 9. ∎

From Lemma 23 we derive that any $\theta \geq 0$, such that Equation (13) does not hold, is an upper bound on the response time of $\tau_{k,\ell}$.

Theorem 24. *Let $\vec{x} = (x_1, \ldots, x_{k-1}) \in \{0, 1\}^{k-1}$ and $a \in \mathbb{N}$. If there exist some $\theta \geq 0$ such that*

$$a \cdot (C_k + S_k) + \sum_{i=1}^{k-1} \left(\begin{array}{l} x_i \cdot A_i^1(\theta + Q_i^{\vec{x}}) \\ +(1 - x_i) A_i^0(\theta + Q_i^{\vec{x}}) \end{array} \right) \leq \theta \quad (14)$$

with $Q_i^{\vec{x}}$ defined as in Lemma 23, then θ is an upper bound on $f_{k,\ell} - r_{k,\ell-a+1}$ for all a-th jobs $\tau_{k,\ell}$ in a suspension-aware busy-interval of τ_k. In particular,

$$R_k^a := \theta - \inf \{\Delta \geq 0 \,|\, \alpha_k^u(\Delta) \geq a\} \quad (15)$$

is an upper bound on the response time of any a-th job in a suspension-aware busy-interval of τ_k.

Proof: Let $\tau_{k,\ell}$ be any a-th job in a suspension-aware busy-interval of τ_k. We prove this theorem by contraposition.

Assume that $\theta \geq 0$ is not an upper bound on $f_{k,\ell} - r_{k,\ell-a+1}$ but Equation (14) holds. In this case, we know that $f_{k,\ell} > \theta + r_{k,\ell-a+1} = \theta + t_k \geq \theta + t_k^*$. In particular $\theta \in [0, f_k - t_k^*)$. Applying Lemma 23 yields that (14) does not hold. This contradicts the assumption.

Since we have shown that $\theta \geq f_{k,\ell} - r_{k,\ell-a+1}$, we conclude

$$\begin{aligned} R_k^a &= \theta - \inf \{\Delta \geq 0 \,|\, \alpha_k^u(\Delta) \geq a\} && (16)\\ &\geq f_{k,\ell} - r_{k,\ell-a+1} - \inf \{\Delta \geq 0 \,|\, \alpha_k^u(\Delta) \geq a\} && (17)\\ &\geq f_{k,\ell} - r_{k,\ell-a+1} - (r_{k,\ell} - r_{k,\ell-a+1}) && (18)\\ &= f_{k,\ell} - r_{k,\ell}, && (19) \end{aligned}$$

i.e., R_k^a is a response time upper bound. Since $\tau_{k,\ell}$ was chosen to be any a-th job in a suspension-aware busy-interval of τ_k, the response time upper bound is valid for all of them. ∎

D. Step 4: The Schedulability Test

To provide a response time upper bound for all jobs of τ_k using Theorem 24, we need to figure out which job in a suspension-aware busy-interval has the highest response time. In this regard, we first need to figure out the maximal number $\tilde{a}$ of jobs that belong to one suspension-aware busy-interval. A sufficient condition for the maximal number is $R_k^{\tilde{a}} \leq \inf \{\Delta \geq 0 \,|\, \alpha_k^u(\Delta) \geq \tilde{a} + 1\} - \inf \{\Delta \geq 0 \,|\, \alpha_k^u(\Delta) \geq \tilde{a}\}$, i.e., the $\tilde{a}$-th job is finished before the next job is released. If the maximal number can be detected with this condition, then a worst-case response time upper bound is provided by the following corollary.

Corollary 25. *Let $\tilde{a} \in \mathbb{N}$ be the lowest natural number such that the response time upper bound $R_k^{\tilde{a}}$ derived by Theorem 24 is at most $\inf \{\Delta \geq 0 \,|\, \alpha_k^u(\Delta) \geq \tilde{a} + 1\} - \inf \{\Delta \geq 0 \,|\, \alpha_k^u(\Delta) \geq \tilde{a}\}$. Then $R_k := \max_{a=1,\ldots,\tilde{a}}(R_k^a)$ is an upper bound on the worst-case response time of τ_k.*

Proof: To prove this corollary, we need to show that each job of τ_k is an a-th job in a suspension-aware busy-interval of τ_k for some $a \in \{1, \ldots, \tilde{a}\}$. We do this by contraposition. Let $\tau_{k,\ell}$ be the first job of τ_k which is not an a-th job in a suspension-aware busy-interval of τ_k with $a \leq \tilde{a}$ according to Definition 2.

Let a be the smallest positive integer, such that $\tau_{k,\ell}$ is the a-th job in a suspension-aware busy-interval of τ_k. Due to our assumption, $a > \tilde{a}$. The job $\tau_{k,\ell-a+\tilde{a}}$ is the $\tilde{a}$-th job in a suspension-aware busy-interval of τ_k. By Theorem 24, $\tau_{k,\ell-a+\tilde{a}}$ finishes no later than at time

$$\begin{aligned} & r_{k,\ell-a+\tilde{a}-\tilde{a}+1} + R_k^{\tilde{a}} + \inf \{\Delta \geq 0 \,|\, \alpha_k^u(\Delta) \geq \tilde{a}\} && (20)\\ & \leq r_{k,\ell-a+1} + \inf \{\Delta \geq 0 \,|\, \alpha_k^u(\Delta) \geq \tilde{a} + 1\} && (21)\\ & \leq r_{k,\ell-a+\tilde{a}+1}. && (22) \end{aligned}$$

In particular, this means that $\tau_{k,\ell-a+\tilde{a}}$ and all previous jobs are finished at time $r_{k,\ell-a+\tilde{a}+1}$. As a result, $\tau_{k,\ell}$ is also an

Algorithm 1 Sufficient schedulability test.

```
for k = 1, ..., n do                                   ▷ Loop tasks.
    for a = 1, 2, 3, ... do
        if a > a_max then
            return False
        comp := inf{Δ ≥ 0 | α_k^u(Δ) ≥ a + 1}
                - inf{Δ ≥ 0 | α_k^u(Δ) ≥ a}
        θ := 0
        while True do
            Compute lhs of Eq. (14).
            if lhs ≤ θ then                            ▷ Result found.
                Break
            else if lhs > D_k then                     ▷ Too high.
                return False
            else                                       ▷ Continue search.
                θ := lhs
        R_k^a := θ - inf{Δ ≥ 0 | α_k^u(Δ) ≥ a}
        if R_k^a ≤ comp then
            ã := a
            Break
    R_k := max_{a=1,...,ã}(R_k^a)                      ▷ WCRT upper bound.
return True
```

$(a-\tilde{a})$-th job in a suspension-aware busy-interval of τ_k which contradicts the minimality of a.

We have proven that each job of τ_k is an a-th job in a suspension-aware busy-interval of τ_k for some $a \in \{1, \dots, \tilde{a}\}$ and we use Theorem 24 to provide response time upper bounds R_k^a for all $\tilde{a}$ cases. ∎

In Section II we have shown that any sub-additive upper arrival curve α_i^u is monotonically increasing for all tasks τ_i. Hence, the left hand side (lhs) of Equation (14) is monotonically increasing with respect to θ. We use this monotonicity to apply similar search strategy as classical time-demand analysis. We start by setting $\theta := 0$ and compute the lhs of Equation (14). Whenever θ is less than the lhs, we set θ to the value of the lhs and compute lhs again. When θ is bigger than or equal to lhs, then θ can be used to compute the response time upper bound as in Theorem 24. The procedure is presented in Algorithm 1. Please note that we artificially set an upper bound a_{max} to exit the algorithm when the number of jobs in a suspension-aware busy-interval is unbounded.

Upper arrival curves are typically modeled as step functions. Under the assumption that the upper arrival curve α_i^u is a step function with minimal step size > 0 for all i, the lhs increases by a certain minimal step size in each iteration as well, until either lhs $\leq \theta$ or lhs $> D_k$. As a result Algorithm 1 is deterministic in such a scenario.

In the following, we show that our method dominates the Compositional Performance Analysis, which is shortly introduced in Section III.

Corollary 26. *The worst-case response time analysis by using Compositional Performance Analysis (CPA) for task τ_k with suspension as computation, i.e., execution time $\alpha_k^u(\Delta) \cdot (C_k + S_k)$ for any interval length of $\Delta \geq 0$, and jitter-based higher-priority preemptive interference of task τ_i with execution time $\alpha_i^u(\Delta + R_i) \cdot (C_i)$, is dominated by the analysis in Corollary 25 when all higher-priority tasks are in $\mathbb{T}_0$.*

Proof: When all tasks are in the set $\mathbb{T}_0$, then Equation (14) simplifies to $a \cdot (C_k + S_k) + \sum_{i=1}^{k-1} A_i^0(\theta) \leq \theta$. However, $A_i^0(\theta)$ is upper bounded by $\alpha_i^u(\Delta + R_i) \cdot C_i$. This is the formula that is used for the a-th job in the busy-interval. In our schedulability test, we increase a, until the subsequent job release of τ_k is outside the busy-interval, i.e., until $R_k^a \leq \inf\{\Delta \geq 0 \,|\, \alpha_k^u(\Delta) \geq \tilde{a} + 1\} - \inf\{\Delta \geq 0 \,|\, \alpha_k^u(\Delta) \geq \tilde{a}\}$. This coincides with the test used in CPA analysis. ∎

V. Schedulability Test for Sporadic Tasks.

In this Section, we derive a schedulability test for sporadic tasks with arbitrary deadlines and $T_k > 0$. We do this by constructing an arrival curve and using the result from the previous section. In the end, we show that by restricting to the constrained-deadline case, our schedulability test coincides with the test in [8], which is limited to constrained-deadline task sets. Hence, in this work we provide a natural extension of their analysis to the generalized case with arbitrary deadlines and arrival curves.

For any **arbitrary-deadline** task τ_i with $T_i > 0$, an upper arrival curve is given by

$$\alpha_i^u = \left(\Delta \mapsto \left\lceil \frac{\max(\Delta, 0)}{T_i} \right\rceil = \begin{cases} \left\lceil \frac{\Delta}{T_i} \right\rceil & \Delta \geq 0 \\ 0 & \Delta < 0 \end{cases}\right). \tag{23}$$

Applying this to A_i^1 and A_i^0 yields

$$A_i^1(\Delta) = \left\lceil \frac{\Delta + \max(R_i - T_i, 0)}{T_i} \right\rceil \cdot C_i \tag{24}$$

$$A_i^0(\Delta) = \min\left(\begin{array}{l} \left\lceil \frac{\Delta + R_i}{T_i} \right\rceil, \\ \left\lceil \frac{\max(\Delta - T_i + R_i - C_i^*, 0)}{T_i} \right\rceil \cdot C_i + C_i^* \end{array} \right) \tag{25}$$

for all $\Delta \geq 0$. Please note that $\Delta + \max(R_i - T_i, 0) \geq 0$ which is why $\max(\cdot, 0)$ is omitted in the numerator of (24). Moreover, we have $C_i^* = \min(\left\lceil \frac{R_i}{T_i} \right\rceil \cdot C_i, R_i)$ and $\inf\{\Delta \geq 0 \,|\, \alpha_i^u(\Delta) \geq a\} = (a-1) \cdot T_i$. This leads to the following upper bound on the worst-case response time.

Theorem 27. *Let $\vec{x} = (x_1, \dots, x_{k-1}) \in \{0,1\}^{k-1}$ and $a \in \mathbb{N}$. If there exist some $\theta \geq 0$ such that*

$$\begin{aligned} & a \cdot (C_k + S_k) \\ & + \sum_{i=1}^{k-1} \left(\begin{array}{l} x_i \cdot \left\lceil \frac{\theta + Q_i^{\vec{x}} + \max(R_i - T_i, 0)}{T_i} \right\rceil \cdot C_i + (1 - x_i) \cdot \\ \min\left(\begin{array}{l} \left\lceil \frac{\theta + Q_i^{\vec{x}} + R_i}{T_i} \right\rceil, \\ \left\lceil \frac{\max(\theta + Q_i^{\vec{x}} - T_i + R_i - C_i^*, 0)}{T_i} \right\rceil \cdot C_i + C_i^* \end{array} \right) \end{array} \right) \\ & \leq \theta \end{aligned} \tag{26}$$

with $Q_i^{\vec{x}}$ defined as in Lemma 23, then θ is an upper bound on $f_{k,\ell} - r_{k,\ell-a+1}$ for all a-th jobs $\tau_{k,\ell}$ in a suspension-aware busy-interval of τ_k. In particular,

$$R_k^a := \theta - (a-1)T_i \tag{27}$$

is an upper bound on the response time of any a-th job in a suspension-aware busy-interval of τ_k.

Proof: This follows from Theorem 24 using the arrival curve from Equation (23), and A_i^1 from Equation (24) and A_i^0 from Equation (25). ■

Similar to Corollary 25, we formulate the following.

Corollary 28. *Let $\tilde{a} \in \mathbb{N}$ be the lowest natural number such that the response time upper bound derived by Theorem 27 is at most T_k. Then $R_k := \max_{a=1,\dots,\tilde{a}}(R_k^a)$ is an upper bound on the worst-case response time of τ_k.*

Proof: The corollary follows from the correctness of Corollary 25 by using that $\inf\{\Delta \geq 0 \,|\, \alpha_k^u(\Delta) \geq \tilde{a}+1\} - \inf\{\Delta \geq 0 \,|\, \alpha_k^u(\Delta) \geq \tilde{a}\} = T_k$. ■

For any **constrained-deadline** task τ_i with $T_i > 0$, if its schedulability is already ensured, i.e., $i < k$ in this work, then $R_i \leq D_i \leq T_i$ is ensured. In such a case $C_i^* = C_i$ and the formulas for A_i^1 and A_i^0 from Equation (24) and (25) are further simplified to

$$A_i^1(\Delta) = \left\lceil \frac{\Delta}{T_i} \right\rceil \cdot C_i \tag{28}$$

$$A_i^0(\Delta) = \left\lceil \frac{\Delta + R_i - C_i}{T_i} \right\rceil \cdot C_i \tag{29}$$

for all $\Delta > 0$. Please note that Equation (29) is obtained from (25) in the following way: For all $\Delta > 0$ we have $\Delta - T_i + R_i - C_i \geq \Delta - T_i > -T_i$. Therefore, we obtain $\left\lceil \frac{\max(\Delta - T_i + R_i - C_i, 0)}{T_i} \right\rceil = \max(\left\lceil \frac{\Delta - T_i + R_i - C_i}{T_i} \right\rceil, 0) = \left\lceil \frac{\Delta - T_i + R_i - C_i}{T_i} \right\rceil$ for all $\Delta > 0$. Moreover,

$$\begin{aligned} A_i^0(\Delta) &= \left\lceil \frac{\Delta - T_i + R_i - C_i}{T_i} \right\rceil \cdot C_i + C_i \\ &= \left(\left\lceil \frac{\Delta - T_i + R_i - C_i}{T_i} \right\rceil + 1 \right) \cdot C_i \\ &= \left\lceil \frac{\Delta + R_i - C_i}{T_i} \right\rceil \cdot C_i. \end{aligned}$$

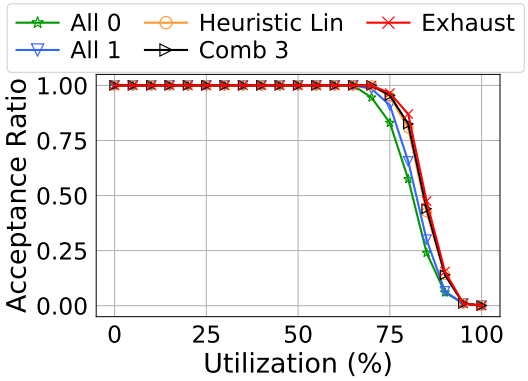
For the constrained-deadline case, we only need to compute the worst-case response time upper bound for $a = 1$, since if $R_i^1 > T_k$ then $R_i^1 > D_k$ as well and the schedulability test fails. The schedulability test formulated with A_i^1 and A_i^0 from Equation (28) and (29) with only $a = 1$ coincides with the schedulability test in [8, Corollary 1]. The high performance of the abovementioned schedulability test demonstrated in [8] indicates high performance of the schedulability test derived in this work.

VI. Choice of $\mathbb{T}_0$ and $\mathbb{T}_1$

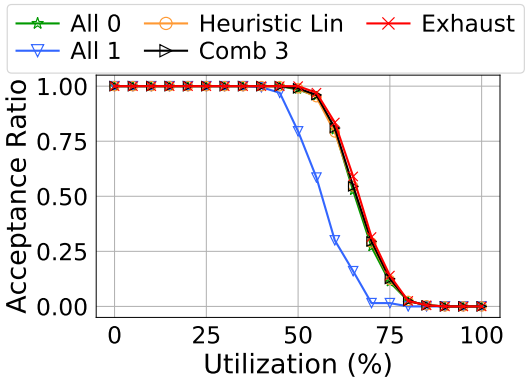
For the schedulability test presented in Section IV any partition of the task set $\mathbb{T}$ into $\mathbb{T}_0$ and $\mathbb{T}_1$ can be used. To fully utilize the power of the proposed test, all possibile combinations of $\mathbb{T}_0$ and $\mathbb{T}_1$ should be explored. However, this results in an exhaustive search of 2^{k-1} different partitions of the $k-1$ higher-priority tasks, for which every schedulability test itself also takes high time complexity.

This limitation was also presented by Chen et al. [8] in their analysis for constrained-deadline task systems. They presented some heuristics, which can also be applied here for arbitrary-deadline task systems and tasks with arrival curves. Here, we shortly present the heuristics that can be used to reduce the complexity significantly. They are compared in the next section. However, those heuristics are not the focus of this paper and should be further examined in the future.

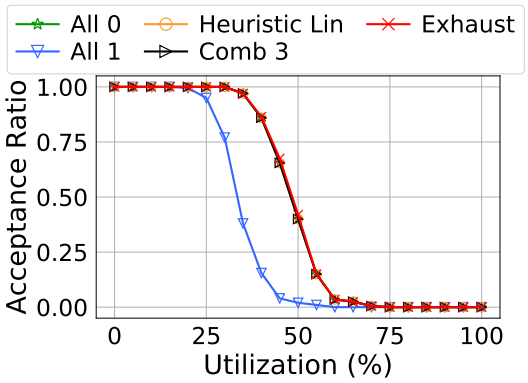
The most simple heuristic is to include all into $\mathbb{T}_0$, i.e., $\mathbb{T}_0 = \mathbb{T}$ and $\mathbb{T}_1 = \emptyset$, or to include all tasks into $\mathbb{T}_1$, i.e., $\mathbb{T}_0 = \emptyset$ and $\mathbb{T}_1 = \mathbb{T}$. In Section VII we observe that such a simple heuristic is already sufficient in many cases.



(a) Low suspension in $[0, 0.1] \cdot (T_i - C_i)$.



(b) Medium suspension in $[0.1, 0.3] \cdot (T_i - C_i)$.



(c) High suspension in $[0.3, 0.5] \cdot (T_i - C_i)$.

Figure 5: Acceptance ratio of our schedulability test with different heuristics as described in Section VI.

As argued in Section V, our analysis is a natural extension to arbitrary-deadline task systems from the analysis for constrained-deadline task systems in [8]. In their work they provide a *linear approximation* that is stated as follows: τ_i is in $\mathbb{T}_1$ if $\frac{C_i}{T_i}(R_i - C_i) > S_i(\sum_{j=1}^{i} \frac{C_j}{T_j})$; otherwise $\tau_i \in \mathbb{T}_0$. They provided mathematical reasoning about this linear approximation.

Our analysis for arbitrary-deadline task systems is much more involved and more difficult to approximate to find a closed-form break-even equation to judge whether it is better to place τ_i in $\mathbb{T}_0$ or in $\mathbb{T}_1$. We do not have concrete mathematical approximations to derive any classification strategies to partition $\mathbb{T}$ into $\mathbb{T}_0$ and $\mathbb{T}_1$. However, the above linear approximation can still be applied. Unfortunately, when considering arrival curves, it is possible that T_i is 0 and $C_i/T_i \to \infty$ for $T_i \to 0$. The above linear approximation is therefore invalid. One possible patch is to apply the linear approximation of the arrival curve by defining $slope_i$ and $const_i$ such that $\alpha_i^u(\Delta) \leq const_i + slope_i\Delta$ holds for all $\Delta \geq 0$ for every task τ_i. Then, we can heuristically put task τ_i in $\mathbb{T}_1$ if $slope_i(R_i - C_i) > S_i(\sum_{j=1}^{i} slope_j)$; otherwise $\tau_i \in \mathbb{T}_0$.

VII. Evaluation

In order to evaluate the performance of our proposed schedulability analysis presented in Section IV, we conduct three different experiments using synthetically generated tasks sets as follows:

1) We demonstrate how the heuristics from Section VI perform for arbitrary-deadline task sets (Experiment – Suspension Time).
2) We show that the increase of deadlines can be exploited in our schedulability test, i.e., the schedulability is increased with increased deadlines (Experiment – Varying Deadline).
3) We examine the performance of our schedulability test for arrival curves obtained from tasks with release jitter (Experiment – Release Jitter).

The source code that is used to conduct the experiments is released on Github [44]. In all our experiments, we use the Algorithm 1 configured with $a_{max} = 10$. The values of $\inf\{\Delta \geq 0 \,|\, \alpha_i^u(\Delta) \geq a\}$ for different a are stored in a list to improve the execution efficiency of the schedulability test. For all three experiments, we present the *acceptance ratio* of the test under analysis, i.e., the number of task sets that are deemed schedulable by the test divided by the total number of task sets. We consider deadline-monotonic scheduling, i.e., the task with a lower relative deadline has a higher priority, and ties are broken arbitrarily. We emphasize that we do not consider a constrained-deadline scenario, since in this case our method is identical to the current state-of-the-art analysis in [8], which dominates all other valid analyses for the studied problem.

A. Experimental Setup and Generation

For each total utilization between 0% and 100% in steps of 5% we randomly generate 200 task sets according to the description below. In a first step, given the required length of task sets, we use the UUniFast [2] algorithm to synthesize task utilizations that add up to a specified cumulative (total) utilization. In a second step, the minimum inter-arrival time for each sporadic task is drawn log-uniformly from the interval $[1, 100]$[ms] as suggested in [13]. Based on the utilization U_i and minimum inter-arrival time T_i, the worst-case execution time is calculated by $C_i := U_i \cdot T_i$ [ms]. The arrival curve of the sporadic tasks is given by $\alpha_i^u(\Delta) = \left\lceil \frac{\Delta}{T_i} \right\rceil$.

In our experiments, we evaluate Algorithm 1 for different partitioning strategies of tasks in $\mathbb{T}$ into $\mathbb{T}_0$ and $\mathbb{T}_1$ as described in Section VI. Namely, these are:

- **(All 0)**: All tasks $\tau_i \in \mathbb{T}$ are put to $\mathbb{T}_0$.
- **(All 1)**: All tasks $\tau_i \in \mathbb{T}$ are put in $\mathbb{T}_1$.
- **(Heuristic Lin)**: The linear heuristic from [8] is adopted.
- **(Comb 3)**: We choose the best partition of the above three heuristics All 0, All1, and Heuristic Lin (in each step).
- **(Exhaust)**: In the exhaustive approach we apply our schedulability test for all 2^{k-1} partitions.

B. Experiments

In this section, we present the details of the experiments and describe the results.

Experiment – Suspension Time: In this experiment, we mostly use the settings described in Section VII-A with only 10 tasks per task set due to the time complexity of (Exhaust). Moreover, we draw the relative deadline D_i uniformly from the interval $[0.8T_i, 1.2T_i]$ and consider three different configurations of self-suspension time:

- *low* – S_i is drawn uniformly from $[0, 0.1](T_i - C_i)$,
- *medium* – S_i is drawn uniformly from $[0.1, 0.3](T_i - C_i)$,
- *high* – S_i is drawn uniformly from $[0.3, 0.5](T_i - C_i)$.

Figure 5 shows the evaluation results, in which (All 0) and (All 1) do not dominate each other. When the suspension time is short, (All 1) is better than (All 0), in Figure 5a. When the suspension time is long, (All 0) is better than (All 1), in Figure 5c . Moreover, (Heuristic Lin) outperforms both of them for all scenarios. The benefit of exhaust is marginal compared to the linear heuristic as can be seen in all Figures 5a, 5b, and 5c.

Experiment – Varying Deadlines: In this experiment, we use the settings described in Section VII-A with 30 tasks per task set and draw the suspension time S_i uniformly at random from the interval $[0, 0.5(T_i - C_i)]$ for each task. In addition, the deadline of all tasks τ_i is set to $D_i = X \cdot T_i$ **(DX)** for $X = 1.0, 1.1, \ldots, 1.5$. We evaluate Algorithm 1 with the partitioning strategy (Comb 3) and show the results in Figure 6. The results show that the acceptance ratio of our algorithm improves with increased deadlines.

Experiment – Release Jitter: In this experiment, we use the settings described in Section VII-A with 10 tasks per task set and draw the suspension time S_i uniformly from the interval

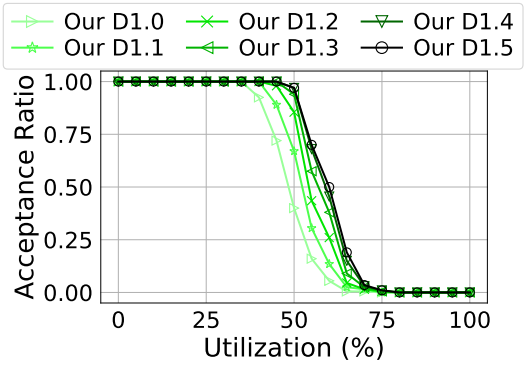


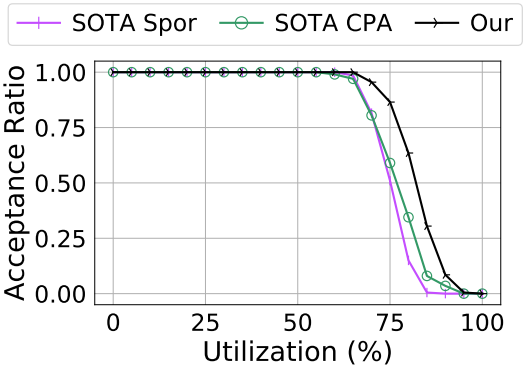
Figure 6: Acceptance ratio of our schedulability test with extended deadlines.

$[0, 0.1](T_i - C_i)$ for each task. The relative deadline of task τ_i is drawn uniformly from the interval $[0.8, 1.2]T_i$. Moreover, we consider task sets with release jitter of 10% or 20%. That is, after the generation of C_i, T_i, D_i, S_i for τ_i, we add $jitter_i$ as $0.1T_i$ or $0.2T_i$. The corresponding arrival curve of τ_i is then $\alpha_i^u(\Delta) = \left\lceil \frac{\Delta + jitter_i}{T_i} \right\rceil$. There are two different state-of-the-art methods that we compare with. For the CPA state of the art **(SOTA CPA)**, we adhere to the state of the art presented in Corollary 26. For the sporadic state of the art **(SOTA Spor)**, we transform the task set into a constrained-deadline task set by reducing the deadline of each τ_i to $\min(D_i, T_i)$ and keeping its original priority. Then we adopt [8] using the heuristic with three partitions, similar to (Comb 3). The schedulability of the transformed task set indicates the schedulability of the original task set. The arrival curve suggests that the minimum inter-arrival time of two consecutive jobs is $T_i - jitter_i$ and we can shorten the relative deadline to $T_i - jitter_i$ if $D_i > T_i - jitter_i$.

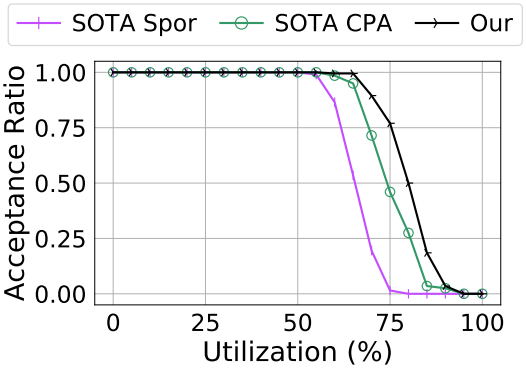
The results for the release jitter of 10% or 20% are shown in Figure 7a and Figure 7b respectively. Our schedulabiltiy test **(Our)** performs much better than the state of the art (SOTA Spor) and (SOTA CPA) for the scenario with release jitter. We note that we also conducted experiments with higher suspension and larger number of tasks per task set as well. In such a case, the suspension time per task becomes longer and the scenario of having tasks in $\mathbb{T}_1$ is less beneficial. When the analysis with all tasks in $\mathbb{T}_0$ is superior, our analysis becomes also jitter-based analysis which is still superior to the (SOTA CPA) but the gain of the acceptance ratio becomes smaller.

VIII. Conclusion

In this paper, we examine uniprocessor fixed-priority preemptive scheduling for self-suspending task sets. More specifically, we present a suspension-aware schedulability test that is applicable to task sets with arbitrary deadlines and whose release properties are specified by arrival curves. To the best of our knowledge we are the first to present such a result. In the evaluation we show that our schedulability test performs well for tasks with increased deadlines.



(a) Release jitter: 10% of T_i.



(b) Release jitter: 20% of T_i.

Figure 7: Acceptance ratio of task sets with release jitter. Our schedulability test and two state of the art are presented.

This paper answers two open problems that have not been tackled in the literature of real-time systems. Firstly, we demonstrate that the suspension-aware busy-interval can be constructed by including the suspension behavior for arbitrary-deadline task systems and *at most one self-suspending job* of every higher-priority task has to be accounted for the analysis during this busy-interval. Secondly, this is the first result exploring the worst-case response time analysis for arrival curves that can be potentially further integrated into the Real-Time Calculus (RTC) or Compositional Performance Analysis (CPA) to empower their capability.

In the future work, we plan to explore mathematical reasoning for heuristics to partition $\mathbb{T}$ into $\mathbb{T}_1$ and $\mathbb{T}_2$ and explore modular performance analysis in RTC or CPA.

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

[3] 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.

[4] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, pages 10190–10195, 2003.

[5] J.-J. Chen. Computational complexity and speedup factors analyses for self-suspending tasks. In *Real-Time Systems Symposium (RTSS)*, pages 327–338, 2016.

[6] J.-J. Chen, T. Hahn, R. Hoeksma, N. Megow, and G. von der Brüggen. Scheduling self-suspending tasks: New and old results. In *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. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, Neil, 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 Systems*, 55(1):144–207, 2019.

[10] 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.

[11] 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, 2016.

[12] Z. Dong, C. Liu, S. Bateni, K.-H. Chen, J.-J. Chen, G. von der Brüggen, and J. Shi. Shared-resource-centric limited preemptive scheduling: A comprehensive study of suspension-based partitioning approaches. In *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, pages 164–176, 2018.

[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] J. C. Fonseca, G. Nelissen, V. Nélis, and L. M. Pinho. Response time analysis of sporadic DAG tasks under partitioned scheduling. In *11th IEEE Symposium on Industrial Embedded Systems, SIES*, pages 290–299, 2016.

[15] N. Guan and W. Yi. Finitary real-time calculus: Efficient performance analysis of distributed embedded systems. In *Proceedings of the IEEE 34th Real-Time Systems Symposium, RTSS*, pages 330–339, 2013.

[16] 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.

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

[18] 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.

[19] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis–the symta/s approach. *IEE Proc.-Computers and Digital Techniques*, (2):148–166, 2005.

[20] R. Hofmann, L. Ahrendts, and R. Ernst. *CPA: Compositional Performance Analysis*, pages 721–751. Springer Netherlands, 2017.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] K. Lakshmanan and R. Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 3–12, 2010.

[27] K. Lampka, S. Bondorf, J. B. Schmitt, N. Guan, and W. Yi. Generalized finitary real-time calculus. In *IEEE Conference on Computer Communications, INFOCOM*, pages 1–9. IEEE, 2017.

[28] J. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001.

[29] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *proceedings Real-Time Systems Symposium (RTSS)*, pages 201–209, Dec 1990.

[30] 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.

[31] 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.

[32] C. Liu and J. Chen. Bursty-interference analysis techniques for analyzing complex real-time task models. In *Real-Time Systems Symposium (RTSS)*, pages 173–183, 2014.

[33] 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.

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

[35] W. Liu, J.-J. Chen, A. Toma, T.-W. Kuo, and Q. Deng. Computation offloading by using timing unreliable components in real-time systems. In *Design Automation Conference (DAC)*, volume 39:1 – 39:6, 2014.

[36] M. Negrean and R. Ernst. Response-time analysis for non-preemptive scheduling in multi-core systems with shared resources. In *7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, pages 191–200, 2012.

[37] 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.

[38] 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.

[39] L. Schönberger, W. Huang, G. von der Brüggen, K. Chen, and 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.

[40] Y. Tang, N. Guan, W. Liu, L. T. X. Phan, and W. Yi. Revisiting GPC and AND connector in real-time calculus. In *2017 IEEE Real-Time Systems Symposium, RTSS*, pages 255–265. IEEE Computer Society, 2017.

[41] Y. Tang, Y. Jiang, and N. Guan. Improving the analysis of GPC in real-time calculus. In N. Guan, J. Katoen, and J. Sun, editors, *Dependable Software Engineering. Theories, Tools, and Applications - 5th International Symposium, SETTA*, volume 11951 of *Lecture Notes in Computer Science*, pages 106–121. Springer, 2019.

[42] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In *Proceedings of the 39th Design Automation Conference, DAC*, pages 880–885. ACM, 2002.

[43] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. *IEEE International Symposium on Circuits and Systems ISCAS*, 4:101–104, 2000.

[44] TU Dortmund LS12. Suspension-aware analysis for arrival curves. https://github.com/tu-dortmund-ls12-rt/arr_curve, 2021.

[45] 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)*, pages 1–9, 2017.