Tighter Worst-Case Response Time Bounds for Jitter-Based Self-Suspension Analysis

Mario Günzel
TU Dortmund University, Germany

Georg von der Brüggen
TU Dortmund University, Germany

Jian-Jia Chen
TU Dortmund University, Germany

Abstract

Tasks are called self-suspending if they can yield their ready state (specifically, releasing the processor while having highest priority) despite being incomplete, for instance, to offload computation to an external device or when waiting on access rights for shared resources or data. This self-suspending behavior requires special treatment when applying analytical results to compute worst-case response time bounds. One typical treatment is modeling self-suspension as release jitter in a so-called jitter-based analysis. The state of the art, when considering task-level fixed-priority scheduling, individually quantifies the jitter term of each higher-priority task by its worst-case response time minus its worst-case execution time.

This work tightens the jitter term by taking the execution behavior of the other higher-priority tasks into account. Our improved jitter-based analysis analytically dominates the previous jitter-based analysis. Moreover, an evaluation for synthetically generated sporadic tasks demonstrates that this jitter term results in tighter worst-case response time bounds for self-suspending tasks. We observe an improvement for up to 55.89% of the tasksets compared to the previous jitter-based analysis.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Computer systems organization → Embedded systems; Software and its engineering → Real-time schedulability

Keywords and phrases Worst-Case Response Time, WCRT, Jitter, Self-Suspension, Analysis

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2024.4

Funding 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).

1 Introduction

In real-time systems, recurrent tasks have to satisfy timing constraints. Specifically, for hard real-time systems, every task instance (called a job) has to finish before its absolute deadline, where the absolute deadline of a job is usually the release time of the job plus the task's relative deadline. Schedulability tests have been developed to provide guarantees that there is no deadline miss. Worst-case response time (WCRT) analyses provide these guarantees by showing that for any job of the task the maximum response time (i.e., the maximum time between a job release and its finishing time) is not larger than the task's relative deadline.

For ordinary sporadic and periodic real-time task systems, in which no job yields its ready state until it completes, the WCRT of a task under preemptive fixed-priority scheduling on uniprocessor systems can be computed using time-demand analysis (TDA) [31, 37]. However, in many real-world applications (e.g., when computation is offloaded to hardware accelerators [21, 41] or when access to the shared resource is denied in multiprocessor locking protocols [8]), a job may yield its ready state between its arrival and completion. In the literature, such behavior (a job yields its ready state before its completion) is referred to as *self-suspension* as a job may *suspend* itself from the ready state and resume at a later point in time. In such cases, TDA does not yield a safe upper bound on the WCRT without further treatment of the suspension time.

Since the observation by Rajkumar et al. [46] in 1988 that self-suspending tasks need a special treatment in the analysis, many results have been provided in the literature. However, since self-suspension can induce several non-trivial phenomena, counterexamples for multiple analyses from before 2014 were found. Details can be found in the review paper by Chen et al. [17], which concludes that due to the self-suspension behavior, "*[...] key insights underpinning the analysis of non-self-suspending tasks no longer hold*" [17].

This paper considers dynamic self-suspending sporadic real-time tasks, one of the two predominately studied models, under preemptive fixed-priority scheduling on a single processor platform. For the dynamic self-suspension model, jobs may suspend arbitrarily often as long as the jobs total suspension time does not exceed the maximum suspension time of the task. The tasks are assumed to have constrained deadlines, that is, their relative deadline is no more than their minimum inter-arrival time. According to the classifications of the review by Chen et al. [17], existing sound results for this scenario quantify the interference from the higher-priority tasks by adopting *suspension-oblivious* analysis, *carry-in* jobs [29, 39], *jitter-based* analysis [6, 29, 43, 45], and *blocking-based* analysis [40, Page 162] [14].

We focus on jitter-based analysis, which has been widely adopted for application scenarios with self-suspensions (e.g., multiprocessor synchronization protocols [7,10,30,36], computation offloading [12], and scheduling of parallel tasks [11, 49]) and is still a common approach to date (e.g., [4]). As long as the release jitter is set correctly, self-suspending tasks can be analyzed as tasks with release jitter, utilizing the related analysis by Tindell et al. [48].

The only known safe release jitter for a higher-priority self-suspending task is setting it to its WCRT minus its worst-case execution time[1] (i.e., $R_i - C_i$ for τ_i in our notation defined in Section 2). Rationale behind this approach is that the execution of the first job (the so-called *carry-in job*) must still finish within the WCRT. Hence, it can only be pushed for at most $R_i - C_i$ time units, assuming that the carry-in job has an execution time of C_i. This seemingly trivial analysis was considered too pessimistic before 2015, when attempts to set the suspension time of a higher-priority task as its release jitter [2, 3, 42] were shown unsafe [6, 17, 43]. Since 2015, the jitter term $R_i - C_i$ has gradually become typical as there has been no analytical improvement, which raises the question:

Is there really no space for improvement?

In this work, we answer this question by providing an improved jitter-based response-time analysis. Our approach is based on the observation that the typical jitter-based analysis is pessimistic, in the sense that for a jitter of $R_i - C_i$ the full workload of the carry-in job must be executed within C_i time units to respect the WCRT bound. Hence, the carry-in job must be executed simultaneously with other higher-priority tasks (details can be found in Section 3). For our approach, we formulate a lower bound $R_i^- \geq C_i$ on the time that is takes to execute the workload of the carry-in job, respecting the interference of other higher-priority tasks on the carry-in job. We show that the jitter-term $R_i - R_i^-$ is safe, thus dominating the previous jitter-based analysis.

[1] Rajkumar [45] and Huang et al. [29] apply a more pessimistic release jitter setting of relative deadline minus worst-case execution time, assuming that a tasks WCRT is no more than its relative deadline.

To achieve the new jitter-term, we modify the typical proof structure of jitter-based analysis. While typical proofs of jitter-based analysis extend the analysis window to the left-hand side, we move the analysis window to the right-hand side. A more detailed discussion on why this treatment is necessary can be found in Section 4.2. To the best of our knowledge, this approach is a novelty for jitter-based analysis and may lead to a series of improved analytical results in jitter-based scenarios.

Our Contributions. In this work, we tighten the WCRT bounds of sporadic self-suspending tasks under uniprocessor fixed-priority scheduling by proposing an improved jitter-based analysis.

- We present our improved jitter-based analysis in Section 4. The proof approach is different from current approaches, since we shift the analysis window *to the right* instead of to the left. We first provide the high-level proof concepts together with a running example in Section 4.1, followed by a detailed and rigorous proof in Section 4.2.
- The dominance over the previous jitter-based analysis is shown in Section 5.
- In Section 6, we demonstrate the improvement of our jitter-based analysis in an evaluation considering synthesized tasksets. Specifically, we show that a tighter worst-case response time can be found in up to 55.89% of the tasksets for the previous jitter-based analysis.

An example for the pessimism of the typical jitter-based analysis is given in Section 3. The system model is specified in Section 2 while related work is discussed in Section 7. In Section 8, we discuss potential improvements for the unifying response-time analysis [16] (a hybrid solution using jitter-based and blocking-based analysis).

2 System Model

Let $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ be a taskset of n sporadically released dynamic self-suspending tasks with constrained deadlines. Each task $\tau_i \in \mathbb{T}$ releases an infinite number of task instances, called jobs. A task is described by a tuple $\tau_i = (C_i, S_i, T_i, D_i)$, where $C_i > 0$ is the worst-case execution time, $S_i \geq 0$ is the maximum suspension time, $T_i > 0$ is the minimum inter-arrival time, and $D_i \leq T_i$ is the (constrained) relative deadline.

Two subsequent job releases of a task τ_i are separated by at least T_i time units. Each job γ released by τ_i has to be executed for a certain amount of time $c_\gamma \in (0, C_i]$. We denote by f_γ the finishing time of γ. During execution, a job may suspend itself; that is, it releases the processor voluntarily and another job can be executed during that time frame. We consider the dynamic self-suspension model, where γ may suspend itself several (but finitely many) times as long as the maximum suspension time S_i is not exceeded. This separates the job γ into a finite number of execution and suspension segments. However, in contrast to the segmented or hybrid self-suspension model, no upper bound on the number of segments is known beforehand.

To comply with the timing constraints of the system, it has to be ensured that the job finishes before (i.e., not later than) its absolute deadline $r_\gamma + D_i$. The response time $f_\gamma - r_\gamma$ of a job is defined as the time between its release r_γ and its finishing time f_γ. The maximum response time among all jobs of task τ_i is the WCRT R_i. We can test whether a task τ_i is schedulable by verifying that its worst-case response time is not greater than its relative deadline D_i.

In this paper, we consider preemptive fixed-priority scheduling on a single processor platform. Each task is assigned a unique priority level and the scheduler always dispatches the job of the highest-priority task among all released jobs that are not self-suspended. We say the system is busy during an interval if at all times during the interval workload is being executed. Due to the self-suspending behavior, there might be no workload executed although there are released but unfinished jobs.

We assume that the task priorities are predefined and that the tasks are numbered in decreasing priority order, that is, τ_1 has the highest priority and τ_n has the lowest priority. We denote by $\mathrm{hp}(i) = \{\tau_1, \ldots, \tau_{i-1}\}$ the set of tasks with higher priority than τ_i. When performing the schedulability analysis of a specific task τ_k we assume that all higher-priority tasks $\mathrm{hp}(k)$ are verified to be schedulable, i.e., $R_i \leq D_i$ for all $i = 1, \ldots, k-1$.

3 Pessimism of Typical Jitter-Based Analysis

Jitter-based analysis over-approximates self-suspending behavior by introducing a release jitter J_i for each task τ_i. The idea is that interference from higher-priority tasks on a job of τ_k can be maximized if the computation of the *first* job of each higher-priority task $\tau_i \in \mathrm{hp}(k)$ is pushed back, such that it finishes at its worst-case response time, while all *subsequent* jobs of τ_i arrive as early as possible (i.e., with minimum inter-arrival time), do not self-suspend, and execute their worst-case execution time. This scenario is depicted in the top schedule in Figure 1. We call jobs (of higher-priority tasks) that are released before but impact the job under analysis *carry-in jobs* (e.g., the first jobs of τ_1 and τ_2 in Figure 1).

It is shown by Nelissen et al. [43] and by Chen et al. [16] that $J_i = R_i - C_i$ is a safe jitter. Therefore, for constrained deadline tasks, the WCRT of a task τ_k is upper bounded by the least non-negative value $R_k \leq D_k$ that fulfills

$$R_k = C_k + S_k + \sum_{\tau_i \in \mathrm{hp}(k)} \left\lceil \frac{R_k + J_i}{T_i} \right\rceil C_i \tag{1}$$

with $J_i = (R_i - C_i)$. The value of R_k can be computed using fixed-point iteration.

However, assuming the jitter $J_i = R_i - C_i$ for each higher-priority task is pessimistic since, in this scenario, *parts of the carry-in may have to be executed in parallel to higher priority workload* to finish within the worst-case response time. The following example demonstrates that behavior.

▶ Example 1. We consider the scenario depicted in Figure 1. The schedule consists of three tasks τ_1, τ_2, and τ_3. Task τ_3 is under analysis with an analysis window of length $t = 16$. The higher-priority tasks are described by the tuples $\tau_1 = (C_1 = 1, S_1 = 3, T_1 = 5, D_1 = 5)$ and $\tau_2 = (C_2 = 9, S_2 = 4, T_2 = 21, D_2 = 21)$. We assume that the worst-case response times of the higher priority tasks are upper bounded by $R_1 = 4$ and $R_2 = 17$. The jitter terms are typically computed as $J_1 = R_1 - C_1 = 3$ and $J_2 = R_2 - C_2 = 8$. This means that, to finish within their worst-case response time, parts of the carry-in workload may have to be executed in parallel to other workload. In the top schedule in Figure 1, there is parallel execution during $[8, 9]$, $[10, 11]$, and $[15, 16]$. When executing this workload sequentially while ensuring that the carry-in jobs finish within their worst-case response time, as in the bottom schedule of Figure 1, J_2 must be reduced to 5. As a result, the second job of τ_2 is moved out of the analysis window and the interference from τ_2 is reduced.

We take the previous example to motivate further examination of the jitter term. In particular, we pursue a jitter J_i that ensures that all higher-priority tasks can meet their deadline by respecting higher-priority interference.

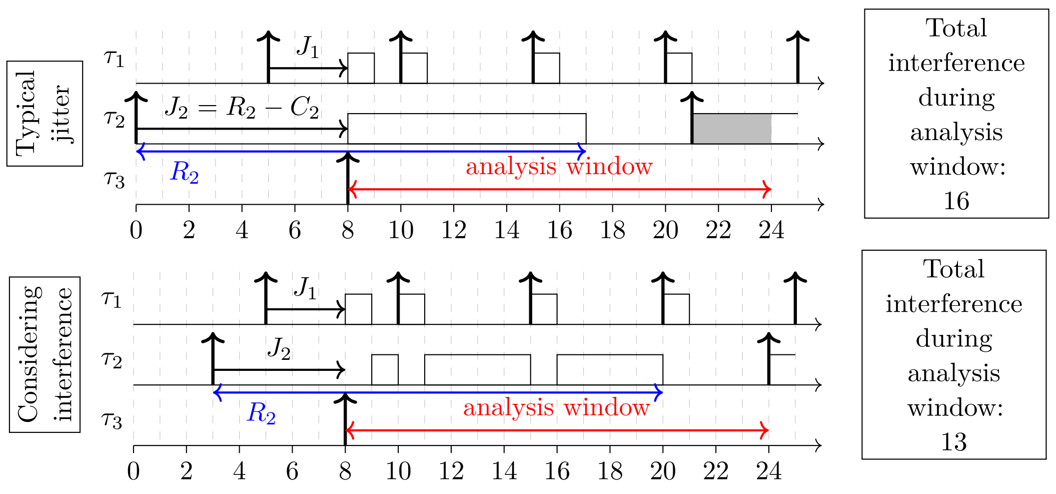


Figure 1 Pessimism of typical jitter-based analysis. We observe that, with the typical jitter, tasks τ_1 and τ_2 have to be executed simultaneously during $[8, 17]$ such that both tasks finish within their worst-case response time ($R_1 = 4, R_2 = 17$). When considering interference from τ_1 on the first job of τ_2, the jitter J_2 has to be reduced such that the first job of τ_2 still finishes within its worst-case response time. In this case, the second job of τ_2 is out of the analysis window.

4 Improved Jitter-Based Analysis

In this section, we show how to include higher-priority interference into the jitter term to achieve a tighter jitter based analysis. Specifically, we propose a reduced jitter term based on the previous observation. Furthermore, we prove that our proposed jitter term is safe, providing the high-level proof concepts together with a running example in Section 4.1, followed by a detailed and rigorous proof in Section 4.2.

To this end, we first revisit the analytical scenario for typical jitter-based analysis discussed in Section 3. Recall that we assume that the task priorities are predefined and that the tasks are numbered in decreasing priority order. Furthermore, when performing the schedulability analysis for τ_k we assume that all higher-priority tasks $\mathrm{hp}(k)$ are verified to be schedulable, i.e., $R_i \leq D_i$ for all $i = 1, \dots, k-1$.

▶ **Definition 2** (Jitter-based analysis scenario.). *The higher-priority jobs released at or after the start of the analysis window fulfill the following analytical properties:*
P1 *The jobs arrive as early as possible (i.e., with minimum inter-arrival time).*
P2 *The jobs do not self-suspend.*
P3 *The jobs execute for their worst-case execution time.*

For the analysis window in the jitter-based analysis scenario, during any interval of length t at least $\left\lfloor \frac{t}{T_j} \right\rfloor$ jobs of any task τ_j are released with a workload of C_j each. Hence, assuming that the carry-in of a task τ_i is the full execution of a job, i.e., C_i time units, it takes at least R_i^- to finish the carry-in, where R_i^- is the least non-negative value[2] that fulfills

$$R_i^- = C_i + \sum_{\tau_j \in \mathrm{hp}(i)} \left\lfloor \frac{R_i^-}{T_j} \right\rfloor \cdot C_j. \tag{2}$$

[2] The value R_i^- can be computed using fixed-point iteration.

To finish within its worst-case response time, the carry-in workload cannot be released after more than $R_i - R_i^-$ time units. This results in a release jitter of

$$J_i = R_i - R_i^- \tag{3}$$

for the jitter-based analysis scenario. In Example 1, depicted in Figure 1, it takes at least $R_2^- = 9 + \lfloor \frac{11}{5} \rfloor = 11$ time units to finish the carry-in of length $C_2 = 9$. Hence, for that example we would obtain a jitter term of $J_2 = R_2 - R_2^- = 17 - 11 = 6$ time units, instead of the typical jitter of $R_2 - C_2 = 17 - 9 = 8$.

Our main result is Theorem 3, which shows that the jitter term $J_i = R_i - R_i^-$ is safe to obtain a valid response time analysis.

▶ **Theorem 3.** *Let the jitter term be $J_i = R_i - R_i^-$. The lowest $0 \leq R_k \leq T_k$ that fulfills*

$$R_k = C_k + S_k + \sum_{\tau_i \in \mathrm{hp}(k)} \left\lceil \frac{R_k + J_i}{T_i} \right\rceil C_i \tag{4}$$

is an upper bound on the response time of task τ_k. If $R_k \leq D_k$, then τ_k meets its deadline.

In the following, we provide a detailed proof of Theorem 3. This proof is done by induction. Specifically, we show that for any fixed-priority preemptive schedule Ψ of the taskset $\mathbb{T}$ the response time upper bound R_k holds for the first ξ jobs of τ_k for $\xi = 0, 1, 2, \ldots$.

While the induction start ($\xi = 0$) is trivial, the induction step ($\xi - 1 \mapsto \xi$) is a three-step process. During the process, the schedule is transformed such that properties P1–P3 from the jitter-based analysis scenario in Definition 2 are fulfilled. For the transformation, careful treatment is required since self-suspending tasks are prone to timing anomalies [17], i.e., increasing the execution of jobs or reducing the inter-arrival time might reduce the response time of other jobs. For our transformation, we thoroughly remove suspension in the schedule when necessary to avoid timing anomalies.

We first describe the high-level ideas of each step in Section 4.1 before providing a detailed proof in Section 4.2. To ease understanding, we provide a running example that illustrates each step of the proof in Figure 2 to Figure 5.

Running example. As depicted in Figure 2, we consider a set of 4 tasks $\tau_1, \ldots, \tau_4$ with the following description:

Task τ_i	τ_1	τ_2	τ_3	τ_4
C_i	1	2	6.5	7
S_i	1.5	2	1.5	2
T_i	4.5	7	17	28

The tasks are ordered according to their priorities, i.e., τ_1 has the highest and τ_4 has the lowest priority. We assume that we analyze the task with the lowest priority in the system[3]; in this case τ_4. A job of task τ_4 released at time 6 is under analysis ($k = 4$, $r_k = 6$) and the analysis window has length $t = 15$. hp(4) consists of the higher-priority tasks τ_1, τ_2, and τ_3.

4.1 High-Level Proof

Let Ψ be a fixed-priority preemptive schedule of the taskset $\mathbb{T}$.

[3] Tasks with a priority lower than the task under analysis can be ignored in the analysis, as they do not affect the schedule of the task under analysis under static-priority scheduling.

Induction start ($\xi = 0$). For $\xi = 0$ we need to show that R_k is a response-time upper bound for all jobs in the empty set $\{\}$. This is trivially satisfied.

Induction step ($\xi - 1 \mapsto \xi$). We assume that R_k is a response-time upper bound for the first $\xi - 1$ jobs of τ_k. Let γ_k be the ξ-th job of task τ_k. Moreover, let r_k be the release time of γ_k and let f_k be the time that γ_k finishes. We consider any $t \geq 0$ such that $r_k + t < f_k$, and we prove that in this case $t < C_k + S_k + \sum_{\tau_i \in \text{hp}(k)} \left\lceil \frac{t+J_i}{T_i} \right\rceil C_i$ holds. Hence, if $t \geq C_k + S_k + \sum_{\tau_i \in \text{hp}(k)} \left\lceil \frac{t+J_i}{T_i} \right\rceil C_i$ then $t \geq f_k - r_k$ and t is a response-time upper bound. Since $t = C_k + S_k + \sum_{\tau_i \in \text{hp}(k)} \left\lceil \frac{t+J_i}{T_i} \right\rceil C_i$ for $t = R_k$, we know that R_k is such a response-time upper bound and Theorem 3 follows directly.

Proof steps. We prove that $t < C_k + S_k + \sum_{\tau_i \in \text{hp}(k)} \left\lceil \frac{t+J_i}{T_i} \right\rceil C_i$ for any t with $f_k - r_k > t \geq 0$. The analyzing process for the analysis window $[r_k, r_k + t]$ has three steps. In the first two steps, we simplify the schedule and revise it to obtain the three properties stated in Definition 2, in both cases ensuring that (i) the worst-case response time of the task under analysis is not decreased while (ii) the worst-case response-time bounds of higher-priority tasks still holds. In the third step, we shift the analysis window to derive a worst-case response-time bound for the task under analysis.

1. **Simplifying the scheduling behavior of τ_k.** In the first step, we remove all jobs of τ_k before and after γ_k, and we make γ_k suspension oblivious. Since all jobs of τ_k before γ_k finish within $R_k \leq T_k$, they all finish before r_k and hence removing them has no impact on schedule within the analysis window $[r_k, r_k + t]$. Jobs of τ_k after γ_k have no impact on the analysis window since they are released after $r_k + t \leq r_k + T_k$. For the job γ_k we replace suspension by additional execution, i.e., we replace γ_k by a job with maximal suspension 0 and execution time $C_k + S_k$. This step is depicted in Figure 3. We show in the detailed proof in Section 4.2, that after this treatment still $r_k + t < f_k$ holds and all jobs of higher-priority tasks still finish within their worst-case response-time bound.
2. **Transforming the schedule into the jitter-based analysis scenario.** In the second step, we transform the schedule such that properties P1, P2, and P3 from Definition 2 are fulfilled. That is, for all jobs of higher-priority tasks released at or after the start of the analysis window, we 1) remove any self-suspension, 2) enlarge the execution time to their worst-case execution time, and 3) move their releases to the earliest possible time. To ensure that the interference of carry-in workload during the analysis interval is not decreased, we additionally remove the suspension from the carry-in after r_k. This step is depicted in Figure 4. We show in the detailed proof in Section 4.2, that after this treatment still $r_k + t < f_k$ holds and all jobs of higher-priority tasks still finish within their worst-case response-time upper bound.
3. **Shifting the analysis window and deriving the bound.** Shifting the analysis window is a typical approach to ensure analytically advantageous properties. In the typical jitter-based analysis, the analysis window is shifted to the beginning of the busy-interval, such that all higher-priority jobs that are executed during the analysis window also start during the analysis window. However, in our case we cannot move the analysis window to the left because otherwise, properties P1-P3 from Definition 2 might be violated. Instead, we move the analysis window to the right to the latest time point b such that γ_k is not executed during the interval $(r_k + t, b)$. This ensures that all jobs of higher priority tasks that are released during the new analysis window $[b - t, b]$ also finish before the end of the analysis window. This step is depicted in Figure 5. We show in

the detailed proof in Section 4.2, that indeed all jobs released during $[b-t, b]$ also finish before b, and we derive the bound $t < C_k + S_k + \sum_{\tau_i \in \mathrm{hp}(k)} \left\lceil \frac{t+J_i}{T_i} \right\rceil C_i$ by analyzing the interference during $[b-t, b]$.

4.2 Detailed Proof of Induction Step

We now provide a detailed proof of the induction step by individually proving all three substeps. To formalize our statements, we denote by

$$\mathrm{exec}_i(a,b) \tag{5}$$

the amount of execution time of jobs of task $\tau_i \in \mathbb{T}$ during the interval $[a,b] \subset \mathbb{R}$, and by

$$\mathrm{susp}_i(a,b) \tag{6}$$

the amount of suspension time of jobs of τ_i during $[a,b]$.

The value $t \geq 0$ is chosen such that $r_k + t < f_k$. Therefore, the time that γ_k is executed during $[r_k, r_k+t]$ must be less than C_k. Moreover, by the induction hypothesis, all jobs of τ_k prior to γ_k finish within the worst-case response-time bound R_k. Since $R_k \leq T_k$, all jobs of τ_k prior to γ_k finish before r_k. We conclude that $\mathrm{exec}_k(r_k, r_k+t) < C_k$ holds. Moreover, during the whole analysis interval $[r_k, r_k+t]$ at all times either higher-priority tasks are executed, or γ_k executes or suspends. Hence, we obtain

$$\begin{aligned} t &\leq \mathrm{exec}_k(r_k, r_k+t) + \mathrm{susp}_k(r_k, r_k+t) + \sum_{\tau_i \in \mathrm{hp}(k)} \mathrm{exec}_i(r_k, r_k+t) \\ &< C_k + S_k + \sum_{\tau_i \in \mathrm{hp}(k)} \mathrm{exec}_i(r_k, r_k+t). \end{aligned} \tag{7}$$

During the three analysis steps we transform the schedule such that properties P1–P3 from Definition 2 are fulfilled which allows bounding $\mathrm{exec}_i(r_k, r_k+t)$ by $\left\lceil \frac{t+J_i}{T_i} \right\rceil C_i$. For the transformations, we need to ensure that the property $r_k + t < f_k$ is preserved, i.e., that Equation (7) still holds.

Step 1: Simplify the scheduling behavior of τ_k. In the first step, we apply the following modifications to the schedule:

(i) We remove all jobs of τ_k before γ_k.
(ii) We remove all jobs of τ_k after γ_k.
(iii) We replace γ_k by a job which executes for $C_k + S_k$ time units and does not suspend.

We show that Step 1 preserves the property $r_k + t < f_k$ (in Lemma 4) and that the worst-case response-time bound of higher-priority tasks still holds (in Lemma 5).

▶ **Lemma 4.** *After Step 1, $r_k + t < f_k$ still holds.*

For each of the three modifications in Step 1, we show that $r_k + t < f_k$ still holds after the modification if it holds before the modification.

Proof. By the induction hypothesis, there is no carry-in from previous jobs of τ_k after r_k. Therefore, removing jobs of τ_k before γ_k has no impact on the schedule of γ_k; thus, $r_k + t < f_k$ still holds after (i).

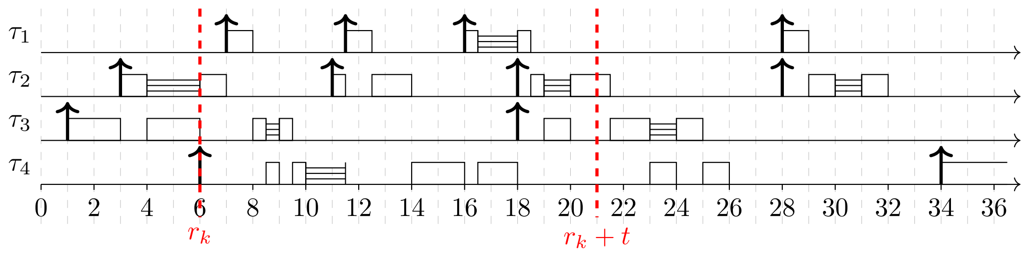


Figure 2 Initial schedule. Tasks may release later than their minimal inter-arrival time, and self-suspension and early completion are allowed.

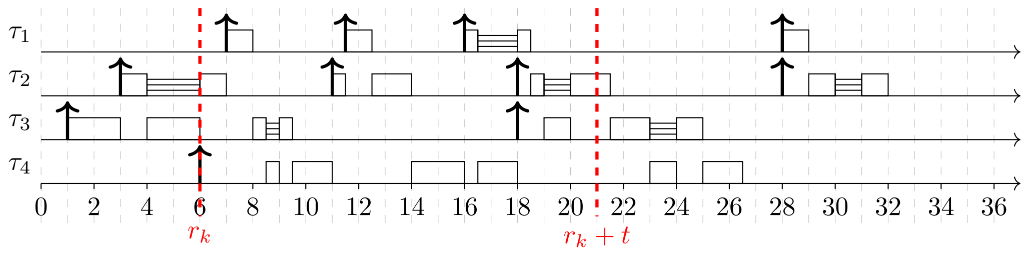


Figure 3 Schedule after Step 1. Job γ_k is made suspension oblivious with execution time $C_k + S_k$, and all other jobs of τ_k are removed.

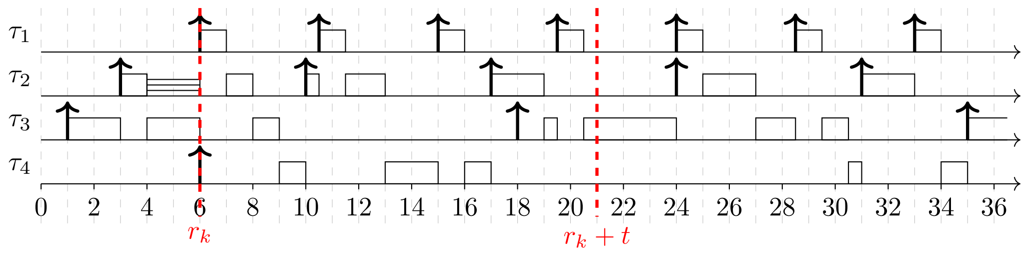


Figure 4 Schedule after Step 2. Higher priority jobs after r_k are released as early as possible, and execute their worst-case execution time without self-suspension. The self-suspension from the carry-in workload is removed. Please note that γ_k continues being executed outside of the displayed time interval, i.e., after 36.

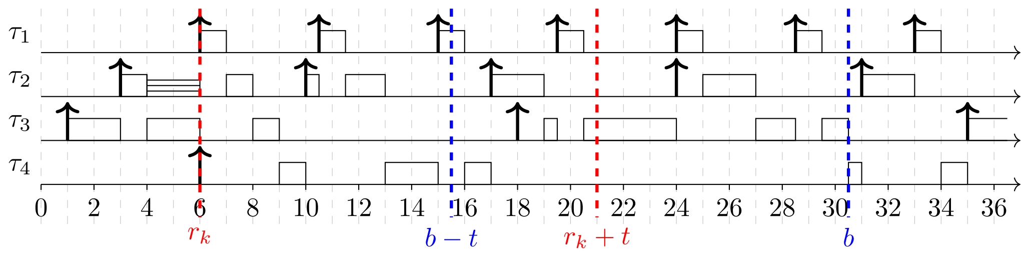


Figure 5 Schedule after Step 3. After $r_k + t = 21$ there is no execution of γ_k before time $b = 30.5$. The shifted analysis window (marked in blue) is $[15.5, 30.5]$. All jobs released during the analysis window finish during the analysis window.

Similarly, jobs of τ_k after γ_k are released no earlier than at time $r_k + t$, and hence have no impact on the schedule during the analysis window. Thus, $r_k + t < f_k$ still holds after (ii).

Modifying the execution and suspension of γ_k has no impact on the higher-priority tasks. Therefore, Step 1 has no impact on the higher-priority interference $\sum_{\tau_i \in \mathrm{hp}(k)} \mathrm{exec}_i(r_k, r_k + t)$, since we assume a preemptive fixed-priority scheduling algorithm. Considering γ_k, replacing an execution time of C_k and suspension time of S_k with execution time of $C_k + S_k$ and without suspension may increase the value calculated in Equation (7). Specifically, we have to account for intervals where the suspension of τ_k overlaps the computation of a higher-priority task. This can be observed when comparing Figure 3 with Figure 2, where after the replacement the interval $[11; 11.5]$, in which γ_k was previously suspended while a job of τ_2 was executed, is added as execution of γ_k during the interval $[26; 26.5]$. However, this may never decrease the workload in a specific interval. Since $t < C_k + S_k + \sum_{\tau_i \in \mathrm{hp}(k)} \mathrm{exec}_i(r_k, r_k + t)$ holds, the execution workload $C_k + S_k$ of γ_k cannot be finished within the analysis window of length t, i.e., $r_k + t < f_k$ still holds after (iii). ◀

▶ **Lemma 5.** *After Step 1, all jobs of higher priority tasks $\tau_i \in \mathrm{hp}(k)$ still finish within their worst-case response-time upper bound R_i.*

Proof. The modification of task τ_k has no impact on the execution of higher priority tasks. Hence, after Step 1 all jobs of higher priority tasks $\tau_i \in \mathrm{hp}(k)$ finish at the same time as before Step 1, and their response time is upper bounded by R_i. ◀

In Step 1, we modified the job γ_k under analysis. In Step 2, our goal is to transform the schedule into the scenario considered in the jitter-based analysis by modifying the execution and releases of higher-priority tasks.

Step 2: Transform the schedule into the jitter-based analysis scenario. In the second step, the following modifications are applied to the schedule:

(i) Remove self-suspension of the carry-in workload.
(ii) Remove self-suspension of jobs of higher priority tasks released at or after r_k.
(iii) Execute jobs of higher priority tasks $\tau_i \in \mathrm{hp}(k)$ released at or after r_k for C_i time units.
(iv) Move the job release of higher-priority tasks to the earliest possible time. That is, a job release $r_\gamma > r_k$ is moved to r_k if there is no previous job release of the same task, or to $\max(r_k, r_{\gamma'} + T_i)$ if there is a previous job release $r_{\gamma'}$ of the same task.

Similar to Step 1, we show that Step 2 preserves the property $r_k + t < f_k$ (in Lemma 6) and that the worst-case response-time bound of higher-priority tasks still holds (in Lemma 7).

▶ **Lemma 6.** *After Step 2, $r_k + t < f_k$ still holds*[4].

Proof. For the sake of readability, we make the following definitions, differentiating the behavior before and after Step 2:

- We denote by f_k^1 the finishing time of γ_k before Step 2, and we denote by f_k^2 the finishing time of γ_k after Step 2.
- We denote by $\mathrm{exec}_i^1(a, b)$ ($\mathrm{exec}_i^2(a, b)$, respectively) the amount of time that jobs of τ_i are executed during the interval $[a, b]$ before (after, respectively) Step 2.
- We denote by $\mathrm{rel}_i^1(a, b)$ ($\mathrm{rel}_i^2(a, b)$, respectively) the number of job releases of τ_i during the interval $[a, b]$ before (after, respectively) Step 2.

[4] In this lemma, f_k refers to the finishing time of job γ_k in the modified schedule (after Step 2).

- We denote by c_i^* the carry-in workload of task τ_i, i.e., the remaining workload at r_k of jobs of τ_i released before r_k. The carry-in workload is the same before and after Step 2, since the schedule before r_k is not modified and the carry-in jobs execution is not modified as well. Only the suspension of carry-in jobs is removed by (i).

In the following, we prove that $r_k + t < f_k^2$. To this end, we assume that $r_k + t \geq f_k^2$ for contradiction.

After Step 2, the carry-in is executed without suspension. Therefore, the carry-in must finish before γ_k can start its execution. Moreover, all jobs of higher priority tasks $\tau_i \in \mathrm{hp}(k)$ released during $[r_k, f_k^2)$ execute without suspension due to (ii). Therefore, they must finish before f_k^2. Because of (iii), they execute their full WCET C_i. We conclude that

$$\mathrm{exec}_i^2(r_k, f_k^2) = c_i^* + \mathrm{rel}_i^2(r_k, f_k^2) \cdot C_i. \tag{8}$$

When we consider the schedule before Step 2, we know that $f_k^1 > r_k + t$, and that therefore $t < C_k + S_k + \sum_{\tau_i \in \mathrm{hp}(k)} \mathrm{exec}_i^1(r_k, r_k + t)$. Moreover, during $[f_k^2, r_k + t]$, the amount of execution of higher priority tasks is upper bounded by $r_k + t - f_k^2$, i.e.,

$$\sum_{\tau_i \in \mathrm{hp}(k)} \mathrm{exec}_i^1(r_k, r_k + t) \leq \sum_{\tau_i \in \mathrm{hp}(k)} \mathrm{exec}_i^1(r_k, f_k^2) + (r_k + t - f_k^2). \tag{9}$$

We conclude that

$$\begin{aligned}
f_k^2 - r_k &= t - (r_k + t - f_k^2) && (10)\\
&< C_k + S_k + \sum_{\tau_i \in \mathrm{hp}(k)} \mathrm{exec}_i^1(r_k, r_k + t) - (r_k + t - f_k^2) && (11)\\
&\overset{(9)}{\leq} C_k + S_k + \sum_{\tau_i \in \mathrm{hp}(k)} \mathrm{exec}_i^1(r_k, f_k^2). && (12)
\end{aligned}$$

Before Step 2, during $[r_k, f_k^2)$, task τ_i can only execute its carry-in c_i^* or jobs that are released during $[r_k, f_k^2)$ for up to C_i time units. As a result, $\mathrm{exec}_i^1(r_k, f_k^2) \leq c_i^* + \mathrm{rel}_i^1(r_k, f_k^2) \cdot C_i$ holds. Moreover, in Step 2, the job releases of τ_i are modified according to (iv). Since job releases after r_k are either not moved or moved to an earlier time, the number of jobs releases during the interval $[r_k, f_k^2)$ cannot be decreased, i.e., $\mathrm{rel}_i^1(r_k, f_k^2) \leq \mathrm{rel}_i^2(r_k, f_k^2)$. We obtain

$$\mathrm{exec}_i^1(r_k, f_k^2) \leq c_i^* + \mathrm{rel}_i^1(r_k, f_k^2) \cdot C_i \leq c_i^* + \mathrm{rel}_i^2(r_k, f_k^2) \cdot C_i = \mathrm{exec}_i^2(r_k, f_k^2) \tag{13}$$

using Equation (8).

By applying Equation (13) to Equation (12), we obtain

$$f_k^2 - r_k < C_k + S_k + \sum_{\tau_i \in \mathrm{hp}(k)} \mathrm{exec}_i^1(r_k, f_k^2) \leq C_k + S_k + \sum_{\tau_i \in \mathrm{hp}(k)} \mathrm{exec}_i^2(r_k, f_k^2). \tag{14}$$

However, since γ_k finishes before f_k^2 after Step 2, $C_k + S_k + \sum_{\tau_i \in \mathrm{hp}(k)} \mathrm{exec}_i^2(r_k, f_k^2) \leq f_k^2 - r_k$ holds which contradicts Equation (14). ◀

▶ **Lemma 7.** *After Step 2, all jobs of higher priority tasks $\tau_i \in \mathrm{hp}(k)$ still finish within their worst-case response time upper bound R_i.*

Proof. After modifications (i)–(iv), still the schedule of tasks $\mathrm{hp}(k)$ complies with the task specification. That is, all jobs of tasks $\tau_i \in \mathrm{hp}(k)$ execute for up to C_i time units, self-suspend for up to S_i time units and release no earlier than with minimum inter-arrival time. Hence, the worst-case response time bound R_i holds for τ_i in the schedule after Step 2. ◀

We now know that Step 1 and Step 2 both preserves the property $r_k + t < f_k$ and that the worst-case response-time bound of higher-priority tasks still holds. What remains is to find the interval with maximum interference for a given t and to show that this interval fulfills the analytical properties P1–P3 from Definition 2.

Step 3: Shift the analysis window and derive the bound. In the third step, we define the time point b as the latest time point such that γ_k is not executed during $(r_k + t, b)$. More formally, $b := \sup\left\{\tilde{b} \geq r_k + t \,\middle|\, \mathrm{exec}_k(r_k + t, \tilde{b}) = 0\right\}$. We move the analysis window from $[r_k, r_k + t]$ to $[b - t, b]$. This has the benefit that all jobs of higher priority tasks released during the analysis interval also finish within the analysis interval.

▶ **Lemma 8.** *All jobs of higher priority tasks $\tau_i \in \mathrm{hp}(k)$ released during $[b-t, b]$ finish during $[b-t, b]$.*

Proof. We prove this lemma by contradiction. To that end, we assume there is a job γ_i of task $\tau_i \in \mathrm{hp}(k)$ released at $r_i \in [b-t, b]$ and finishing at $f_i > b$. In that case, during $[b, f_i)$ only γ_i and jobs of $\mathrm{hp}(i)$ would be executed. In particular, $\mathrm{exec}_k(b, f_i) = 0$. Therefore, $\mathrm{exec}_k(r_k + t, f_i) = \mathrm{exec}_k(r_k + t, b) + \mathrm{exec}_k(b, f_i) = 0$, which contradicts that b is the latest time point with $\mathrm{exec}_k(r_k + t, b) = 0$. ◀

Moreover, the analysis window fulfills the analytical properties from Definition 2.

▶ **Lemma 9.** *All higher-priority jobs released at or after $b - t$ fulfill the analytical properties P1–P3 from Definition 2.*

Proof. P1–P3 are ensured for all jobs released at or after r_k by transformations (ii), (iii) and (iv) in Step 2. Since $b - t \geq r_k$ by definition of b, P1–P3 also hold for all jobs released at or after $b - t$. ◀

In the following we utilize the properties from Lemma 8 and Lemma 9 to show that $t < C_k + S_k + \sum_{\tau_i \in \mathrm{hp}(k)} \left\lceil \frac{t+J_i}{T_i} \right\rceil C_i$ with $J_i = R_i - R_i^-$. First, we upper bound the execution time in the interval of length t for each task.

▶ **Lemma 10.** *For each $\tau_i \in \mathrm{hp}(k)$, the amount of execution time during $[b-t, b]$ is upper bounded by*

$$\mathrm{exec}_i(b - t, b) \leq \left\lceil \frac{t + R_i - R_i^-}{T_i} \right\rceil C_i. \tag{15}$$

Proof. For $t = 0$, Equation (15) is fulfilled since $\mathrm{exec}_i(b, b) = 0$ and $\left\lceil \frac{R_i - R_i^-}{T_i} \right\rceil C_i \geq 0$. For $t > 0$ we distinguish two cases:

Case 1. At most one job of τ_i is executed during $[b-t, b]$. In that case $\mathrm{exec}_i(b-t, b) \leq C_i \leq \left\lceil \frac{t+R_i-R_i^-}{T_i} \right\rceil C_i$.

Case 2. At least two jobs of τ_i are executed during $[b-t, b]$. Let γ_i be the last job of τ_i that is executed during $[b-t, b]$. The job γ_i must be released after the previous job finishes, i.e., after $b - t$. Hence, $r_{\gamma_i} > b - t \geq r_k$. Due to P3 in Lemma 9, γ_i executes for exactly C_i time units.

Additionally, during $[b-t, b]$ the jobs of $\tau_j \in \text{hp}(i)$ are released with minimum inter-arrival time due to P1 (as shown in Lemma 9). Therefore, during $[r_{\gamma_i}, b] \subseteq [b-t, b]$ at least $\left\lfloor \frac{b-r_{\gamma_i}}{T_j} \right\rfloor$ jobs of $\tau_j \in \text{hp}(i)$ are released. Since P3 holds according to Lemma 9, each of these jobs executes for C_j time units, and due to Lemma 8 each of these jobs finishes before b. Thus, during $[r_{\gamma_i}, b]$ at least C_i and $\left\lfloor \frac{b-r_{\gamma_i}}{T_j} \right\rfloor \cdot C_j$ for all $\tau_j \in \text{hp}(i)$ are executed, i.e.,

$$b - r_{\gamma_i} \geq C_i + \sum_{\tau_j \in \text{hp}(i)} \left\lfloor \frac{b - r_{\gamma_i}}{T_j} \right\rfloor \cdot C_j \tag{16}$$

holds. The value R_i^- is the lowest non-negative value with $R_i^- \geq C_i + \sum_{\tau_j \in \text{hp}(i)} \left\lfloor \frac{R_i^-}{T_j} \right\rfloor \cdot C_j$. Hence, $b - r_j \geq R_i^-$ holds. We conclude that only jobs of τ_i that are released before $b - R_i^-$ are executed during $[b-t, b]$. Furthermore, we know that all jobs of τ_i released before $b - t - R_i$ finish before $b - t$. Therefore, only jobs released after $b - t - R_i$ can be executed during $[b-t, b]$.

To conclude, only jobs released during the half-opened interval $(b - t - R_i, b - R_i^-]$ can be executed during $[b-t, b]$. The maximal number of jobs of τ_i released during $(b - t - R_i, b - R_i^-]$ is $\left\lceil \frac{b-R_i^- -(b-t-R_i)}{T_i} \right\rceil = \left\lceil \frac{t+R_i-R_i^-}{T_i} \right\rceil$. Each of these can be executed for at most C_i time units, i.e., $\text{exec}_i(b-t, b) \leq \left\lceil \frac{t+R_i-R_i^-}{T_i} \right\rceil C_i$, which concludes the proof. ◀

Finally, we show that the inequality $t < C_k + S_k + \sum_{\tau_i \in \text{hp}(k)} \left\lceil \frac{t+J_i}{T_i} \right\rceil C_i$ holds for our derived jitter term $J_i = R_i - R_i^-$ if $r_k + t < f_k$ holds before the transformations.

▶ **Lemma 11.** *If $r_k + t < f_k$ before Step 1, then the inequality $t < C_k + S_k + \sum_{\tau_i \in \text{hp}(k)} \left\lceil \frac{t+J_i}{T_i} \right\rceil C_i$ with $J_i = R_i - R_i^-$ holds.*

Proof. If $r_k + t < f_k$ before Step 1, then Lemma 4 and Lemma 6 ensure that $r_k + t < f_k$ after the transformations of Step 1 and Step 2. Since $r_k + t < f_k$ after Step 2, we have

$$t < C_k + S_k + \sum_{\tau_i \in \text{hp}(k)} \text{exec}_i(r_k, r_k + t). \tag{17}$$

By definition of b, during $[r_k + t, b)$ the processor is busy executing workload of jobs of $\text{hp}(k)$. Therefore, we obtain

$$\sum_{\tau_i \in \text{hp}(k)} \text{exec}_i(r_k, r_k + t) + (b - r_k - t) = \sum_{\tau_i \in \text{hp}(k)} \text{exec}_i(r_k, b) \tag{18}$$

$$= \sum_{\tau_i \in \text{hp}(k)} \text{exec}_i(b - t, b) + \sum_{\tau_i \in \text{hp}(k)} \text{exec}_i(r_k, b - t) \tag{19}$$

$$\leq \sum_{\tau_i \in \text{hp}(k)} \text{exec}_i(b - t, b) + (b - t - r_k). \tag{20}$$

Hence, $\sum_{\tau_i \in \text{hp}(k)} \text{exec}_i(r_k, r_k + t) \leq \sum_{\tau_i \in \text{hp}(k)} \text{exec}_i(b - t, b)$ and

$$t < C_k + S_k + \sum_{\tau_i \in \text{hp}(k)} \text{exec}_i(b - t, b). \tag{21}$$

Using Lemma 10, we achieve $t < C_k + S_k + \sum_{\tau_i \in \text{hp}(k)} \left\lceil \frac{t+J_i}{T_i} \right\rceil C_i$ with $J_i = R_i - R_i^-$. ◀

This concludes the proof of Theorem 3: Since $R_k \geq C_k + S_k + \sum_{\tau_i \in \text{hp}(k)} \left\lceil \frac{R_k + J_i}{T_i} \right\rceil C_i$, we know that $R_k + r_k \geq f_k$ by Lemma 11, i.e., R_k is an upper bound on the response time of γ_k. This was to be shown in the induction step.

5 Dominance

In the preceding section we show that $J_i = R_i - R_i^-$ is a safe jitter for jitter-based worst-case response time analysis of self-suspending tasks. In this section we discuss the dominance of the new jitter-based analysis with $J_i = R_i - R_i^-$ over the previous jitter-based analysis with $J_i = R_i - C_i$. First, we observe that $R_i^- \geq C_i$.

▶ **Lemma 12.** *The worst-case execution time C_i of τ_i is upper bounded by R_i^-.*

Proof. R_i^- is defined by Equation (2) as the least non-negative value that fulfills $R_i^- = C_i + \sum_{\tau_j \in \text{hp}(i)} \left\lfloor \frac{R_i^-}{T_j} \right\rfloor \cdot C_j$. Since $\sum_{\tau_j \in \text{hp}(i)} \left\lfloor \frac{R_i^-}{T_j} \right\rfloor \cdot C_j \geq 0$, we conclude that $R_i^- \geq C_i$. ◀

As a consequence, the new jitter term $R_i - R_i^-$ is upper bounded by the previous jitter term $R_i - C_i$. This results in a dominance relation.

▶ **Theorem 13.** *The proposed jitter-based response time analysis with jitter term $J_i = R_i - R_i^-$ analytically dominates previous jitter-based analysis with jitter term $J_i = R_i - C_i$.*

Proof. We show that whenever the previous jitter-based analysis provides a response time upper bound R_k^{prev}, our jitter-based analysis provides a response time upper bound R_k^{our} as well, and $R_k^{our} \leq R_k^{prev}$. Let $R_k^{prev} \leq D_k$ be the lowest non-negative value that fulfills Equation (1), i.e., $R_k^{prev} = C_k + S_k + \sum_{\tau_i \in \text{hp}(k)} \left\lceil \frac{R_k^{prev} + R_i - C_i}{T_i} \right\rceil C_i$. By Lemma 12, $R_i - C_i \geq R_i - R_i^-$. Therefore, we obtain

$$R_k^{prev} \geq C_k + S_k + \sum_{\tau_i \in \text{hp}(k)} \left\lceil \frac{R_k^{prev} + R_i - R_i^-}{T_i} \right\rceil C_i. \tag{22}$$

We define $f := (t \mapsto t - \sum_{\tau_i \in \text{hp}(k)} \left\lceil \frac{t + R_i - R_i^-}{T_i} \right\rceil C_i)$. That is, f is a combination of a linear part t and a right continuous step function $\sum_{\tau_i \in \text{hp}(k)} \left\lceil \frac{t + R_i - R_i^-}{T_i} \right\rceil C_i$. Hence, f is right continuous, and f grows linearly if the step function stagnates. If the step function jumps, f jumps downward. In particular, $f(t - \Delta) \geq f(t) - \Delta$ for all $\Delta \geq 0$. Moreover, $f(R_k^{prev}) \geq C_k + S_k$ by Equation (22), and $f(0) \leq 0$.[5]

We define $t_0 := \inf \{t \geq 0 \mid f(t) \geq C_k + S_k\}$. Since f is right continuous, $f(t_0) \geq C_k + S_k$. Moreover, $0 < t_0 \leq R_k^{prev}$. We show that $f(t_0) = C_k + S_k$ by contradiction: Assume that $f(t_0) > C_k + S_k$. Define $\Delta = \min(t_0, f(t_0) - (C_k + S_k))$. Then $f(t_0 - \Delta) \geq f(t_0) - \Delta \geq C_k + S_k$ which contradicts the definition of t_0.

We conclude that there exists a $t_0 \in [0, R_k^{prev}]$ such that $f(t_0) = C_k + S_k$. This means that $t_0 = C_k + S_k + \sum_{\tau_i \in \text{hp}(k)} \left\lceil \frac{t_0 + R_i - R_i^-}{T_i} \right\rceil C_i$ for some $t_0 \in [0, R_k^{prev}]$, i.e., $R_k^{our} \leq R_k^{prev}$. ◀

[5] We omit the proof that $R_i^- \leq R_i$ holds, which is necessary to ensure $f(0) \leq 0$, due to space limitations.

6 Evaluation

We evaluated the impact of the improved jitter term on jitter-based analysis considering synthesized tasksets. Specifically, we examined whether the response time of (at least) one of the tasks in the set could be reduced when applying the improved analysis compared to the typical analysis. We did not evaluate the acceptance ratio (i.e., the number of feasible tasksets) for the following two reasons: (i) our response time analysis is not limited to any specific deadline as long as the worst-case response time is no more than the minimum inter-arrival time; (ii) the experimental setting for the configuration of relative deadlines is a decisive factor for demonstrating the improvements of the acceptance ratio. On the one hand, any improvement might potentially move the WCRT bound below the deadline constraint, making the task schedulable. On the other hand, even a large improvement on the WCRT bound might not affect the schedulability at all. Hence, choosing a particular deadline hides the impact on the analytical performance. Therefore, since this work is about improving the WCRT bound, we examined the bound directly.

Evaluation setup. We generated 10 000 tasksets for each combination of the following parameter values:

- number of tasks per set $n \in \{10, 20, 40, 60, 80, 100\}$,
- combined utilization of *execution and suspension* $U_{C+S} \in \{1.0, 2.0, 3.0\}$,
- *execution segments only* utilization $U_C \in \{0.05, 0.1, 0.15, \dots, 0.95\}$, and
- period range $P_{range} \in \{[1, 10], [1, 100], [1, 1000]\}$.

For a specific combination of n, U_{C+S}, U_C, and P_{range}, each set was generated as follows:

1. We generated a utilization vector for the combined utilization of *execution and suspension* using the Dirichlet-Rescale (DRS) [22] algorithm. Specifically, we generated n values $U^i_{C+S}, i = 1, \dots, n$ such that $0 \leq U^i_{C+S} \leq 1$ and $\sum_{i=1}^n U^i_{C+S} = U_{C+S}$.
2. We generated a utilization vector for *execution segments only* using DRS. In particular, n values $U^i_C, i = 1, \dots, n$ with $0 \leq U^i_C \leq U^i_{C+S}$ and $\sum_{i=1}^n U^i_C = U_C$ were generated.
3. Each task minimum inter-arrival time $T_i, i = 1, \dots, n$ was drawn log-uniformly from the interval P_{range}.
4. We derived worst-case execution time C_i and maximum suspension time S_i by calculating $C_i = T_i \cdot U^i_C$ and $S_i = T_i \cdot U^i_{C+S} - C_i$.
5. The tasks were prioritized in Rate-Monotonic (RM) ordering, i.e., the task with the shortest minimum inter-arrival time had the highest priority.

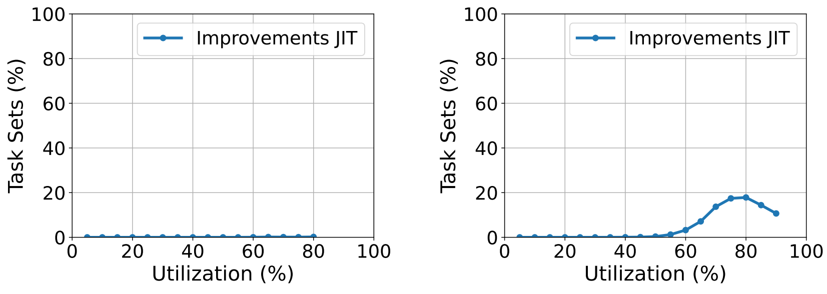
For each generated task τ_k, we consider the following analytical worst-case response time *lower bound* (**LB**):

$$R_k = C_k + S_k + \sum_{\tau_i \in \mathrm{hp}(k)} \left\lceil \frac{R_k + S_i}{T_i} \right\rceil C_i \tag{23}$$

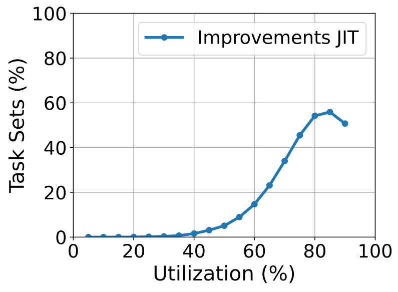
This is the exact response time of a special scenario where each higher-priority task delays the first job for S_i time units, and, therefore, it is a lower bound for the general case. If for any task the lower bound exceeded T_k, then the taskset was discarded and a new one was generated. If the 10 000 tasksets could not be generated after 100 000 tries, the configuration was discarded and omitted from the evaluation.

We applied the following response time analyses for each task:

- **JIT-TYP:** The typical jitter-based worst-case response time (WCRT) analysis from Equation (1) with jitter $J_i = R_i - C_i$.
- **JIT-IMP:** Our improved jitter-based analysis from Theorem 3 with jitter $J_i = R_i - R_i^-$.



(a) $P_{range} = [1, 10]$. **(b)** $P_{range} = [1, 100]$.



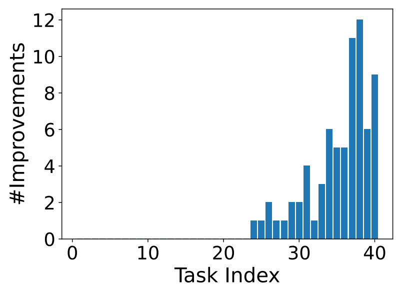
(c) $P_{range} = [1, 1000]$.

Figure 6 Number of improved tasksets in % when using **JIT-IMP** instead of **JIT-TYP**.

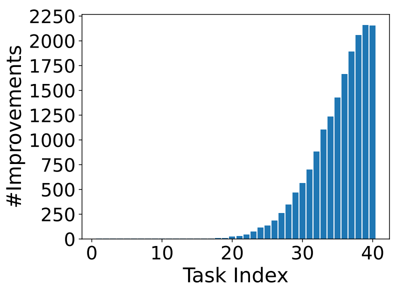
Evaluation results. We evaluated the number of tasksets where the analytical WCRT of at least one task in the set can be improved by utilizing **JIT-IMP** instead of **JIT-TYP** (denoted as **Improvements JIT**). We observed that the amount of suspension and the number of tasks per set have rather low impact on the number of tasksets where the analytical WCRT was improved. Therefore, we only show the results for $U_{C+S} = 2.0$ and $n = 40$ in Figure 6 as the results for other configurations are similar.

The number of tasksets where our improved analysis results in improved WCRTs increases (i) with the period range, and (ii) with the utilization. Specifically, for the period range, we observed barely any improvement for $P_{range} = [1, 10]$, but the WCRT analysis was improved for $P_{range} = [1, 100]$ up to 17.84% and 12.25%, and for $P_{range} = [1, 1000]$ up to 55.89% and 43.51% of the tasksets for the improved jitter-based and the improved unified analysis, respectively. This suggests good potential improvement in practical scenarios where the periods usually range over two to three orders of magnitude [9] and where a period range from 1 to 1000 is common in automotive systems [34].

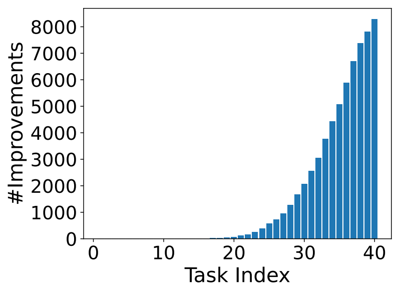
The improvement increases with the utilization up to around 80% utilization and slightly decreases afterwards. Thus, the improvement is most prominent in the area where tasksets are neither most likely anyway schedulable due to low utilization nor anyway not schedulable due to high utilization. Specifically, regarding high utilization tasksets, note that there are no results for utilization values $U_C = 0.95$ in Figure 6 since we could not generate 10 000 tasksets which could potentially be schedulable according to the lower bound **LB** in 100 000 tries. Furthermore, the drop for higher utilization in Figure 6 is likely the result of tasksets which may still be schedulable according to the lower bound **LB** which actually are not schedulable and thus no improvement can be obtained.



(a) $P_{range} = [1, 10]$.



(b) $P_{range} = [1, 100]$.



(c) $P_{range} = [1, 1000]$.

Figure 7 Number of improved tasks when using **JIT-IMP** instead of **JIT-TYP**. The x-axis shows the index of the task in its taskset.

Additionally, we depict the number of improved tasks plotted over the task index (i.e., the priority of that task in its tasksets) in Figure 7. We observe that our improvement has a larger impact for the tasks with lower priority in a taskset. That is, **JIT-TYP** usually has optimal performance for the highest priority tasks and show a larger degree of over-approximation for the lower priority tasks. These lower-priority tasks can benefit from the improved analysis presented in this work. Please note that the number of improved tasks differs significantly for the different evaluation scenarios, which is why we chose to utilize different ranges for the y-axes of the plots.

7 Related Work

In 1988, Rajkumar et al. [46] observed that self-suspending tasks need a special treatment in the analysis. Early results tried to deal with self-suspension by extending classical schedulability analyses and WCRT analyses of non-self-suspending tasks. However, self-suspension can induce several non-trivial phenomena, as discussed in the review paper by Chen et al. [17]. Hence, careful treatment of self-suspending tasks is required. After the first counterexample of the extension of the critical instant theorem in [35] was found by Nelissen et al. [43], several counterexamples of analyses before 2014 have been provided by Chen et al. [17], Bletsas et al. [6], and Günzel and Chen [23,24].

In the literature, the dynamic self-suspension model [1,2,16,19,23,26,27,29,39] and the segmented self-suspension model [5,13,15,24,28,33,43,44,47] are predominately studied. They differ in their specifications of suspension patterns. More specifically, in the segmented self-suspension model, computation and suspension segments of a task are predefined to appear in an interleaved manner, each with bounded duration. On the other hand, in the dynamic self-suspension model, any arbitrary computation and suspension sequence is admissible as long as the worst-case execution time and the maximum suspension time are both respected. The dynamic self-suspension model is classified as a *behavior relaxation* [50] of the segmented self-suspension model. To bridge the gap between the dynamic and the segmented self-suspension model, hybrid self-suspension models have been provided by von der Brüggen et al. [51], assuming that the maximum number of suspension intervals is known but concrete execution/suspension patterns are unknown. Literature reviews for self-suspension have been provided by Chen et al. [17, 18].

This paper considers dynamic self-suspending real-time tasks, which has been examined in multiple papers [6,16,27,29,39] and in the book by Jane Liu [40, Page 162]. Other results [2,32] have been disproved (c.f. the review by Chen et al. [17]). Valid analytical results quantify the interference from the higher-priority tasks by adopting *suspension-oblivious* analysis, *carry-in* jobs [29, 39], *jitter-based* analysis [6, 29, 43, 45], and *blocking-based* analysis [40, Page 162] [14]. Chen et al. [16] show that the *jitter-based* and *blocking-based* analyses do not dominate each other and provide a hybrid solution, called unifying response-time analysis. Günzel et al. [25] extend the unifying response-time analysis to arbitrary deadlines and arrival curves. For EDF scheduling, the result by Devi [19] has been disproved [23]. Further analytical results are provided by Liu and Anderson [38] and Dong and Liu [20] for global EDF and by Günzel et al. [26] for uniprocessor systems. Moreover, Günzel et al. [27] consider EDF-Like scheduling and show that their analysis can be applied for uniprocessor fixed-priority preemptive scheduling. However, for the fixed-priority setup, they achieve worse schedulability, compared to the jitter-based and the unifying analysis.

Regarding non-preemptive scheduling, to the best of our knowledge only two schedulability analyses exist. Casini et al. [11] analyze self-suspending tasks scheduled by partitioned fixed-priority scheduling, while Yalcinkaya et al. [52] consider global multiprocessor scheduling. The latter provide an exact analysis utilizing the UPPAAL model checker.

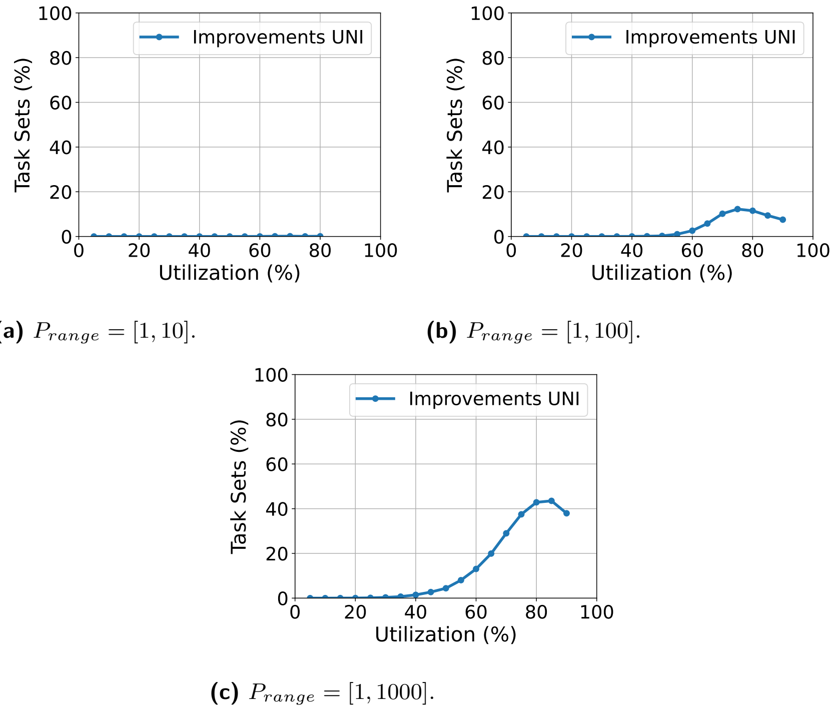
8 Remarks on Improving the Unifying Response Time Analysis

Although this is the first work tightening the jitter term J_i in jitter-based analysis, there is another dominating result on the typical jitter-based analysis in the literature. More specifically, the unifying response time analysis by Chen et al. [16] is a hybrid solution of jitter-based and blocking-based analysis. They show that, for any arbitrary vector assignment $\vec{x} = (x_1, \ldots, x_{k-1}) \in \{0,1\}^{k-1}$, the least non-negative value R_k that fulfills

$$R_k \geq C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{R_k + Q_i^{\vec{x}} + (1-x_i)(R_i - C_i)}{T_i} \right\rceil C_i, \tag{24}$$

with $Q_i^{\vec{x}} = \sum_{j=i}^{k-1} x_j S_j$, is an upper bound on the worst-case response time of τ_k.

When choosing the vector assignment $\vec{x} = (0, \ldots, 0)$, the unifying analysis simplifies to the typical jitter-based analysis, i.e., Equation (1). Hence, it seems intuitive that $(R_i - C_i)$ in Equation (24) can be replaced by the new jitter term $R_i - R_i^-$. However, the underlying analysis structure is fundamentally different. More specifically, the proof of the unifying response-time analysis is based on shifting the analysis window to the *left*, combined with job modifications. The proof of the new jitter term is based on shifting the analysis window to the *right*, combined with different job modifications. Thus, it is not clear if the unifying response-time analysis can utilize the improved jitter term presented in this work.



(a) $P_{range} = [1, 10]$. **(b)** $P_{range} = [1, 100]$. **(c)** $P_{range} = [1, 1000]$.

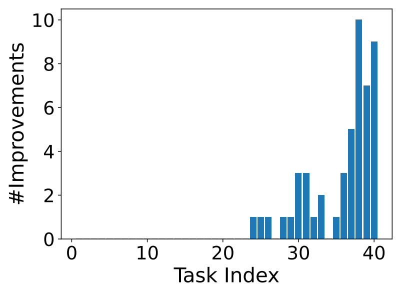
Figure 8 Number of improved tasksets in % when using **UNI-IMP** instead of **UNI-TYP**.

However, to showcase the potential improvement that might be achieved from integrating the improved jitter term into the unifying response-time analysis, we consider the following improvement strategy: First, we compare the result obtained by the unifying response-time analysis with a worst-case response-time lower bound in Equation (23). If the worst-case response-time upper bound from the unifying response-time analysis (Equation (24)) *coincides* with the lower bound (Equation (23)), it is an exact upper bound and there is no further space for improvement. Otherwise, we apply the improved jitter-based analysis (Theorem 3) and take the minimum of both worst-case response-time upper bounds.

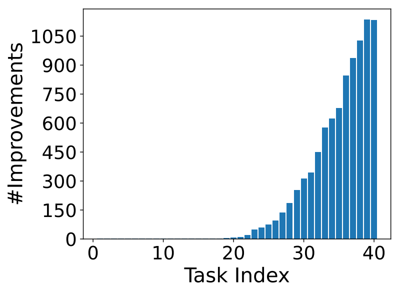
We examined this improvement that can by achieved by this ad-hoc integration, by evaluating a similar scenario as in Section 6. Instead of **JIT-TYP** and **JIT-IMP**, we utilized the following response time analyses for each task:

- **UNI-TYP:** The unifying response time analysis from Chen et al. [16] with their three suggestions for vector $\vec{x}$.
- **UNI-IMP:** The improved unifying response-time analysis using our improved jitter-based analysis as discussed in this Section.

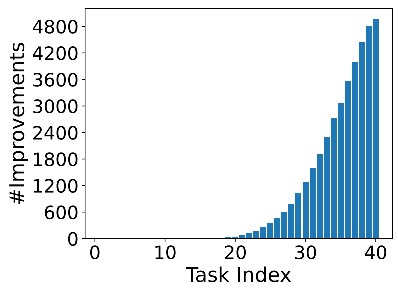
We evaluated the number of tasksets where the analytical WCRT of at least one task in the set can be improved by utilizing **UNI-IMP** instead of **UNI-TYP** (denoted as **Improvements UNI**). As in Section 6, we only show the results for $U_{C+S} = 2.0$ and $n = 40$, as we observed that the amount of suspension and the number of tasks per set have a rather low impact on the number of tasksets where the analytical WCRT was improved. Figure 8 shows the number of improved tasksets and Figure 9 shows the number of improved tasks. We observe that the number of improvements for the typical jitter-based response-time analysis (in Figures 6 and 7) is larger than for the unifying response-time analysis. The reason is that the unifying response time analysis performs already closer to an optimal analysis than



(a) $P_{range} = [1, 10]$.



(b) $P_{range} = [1, 100]$.



(c) $P_{range} = [1, 1000]$.

Figure 9 Number of improved tasks when using **UNI-IMP** instead of **UNI-TYP**. The x-axis shows the index of the task in its taskset.

the jitter-based response-time analysis. However, we observe that there is still a significant amount of improvements for the unifying response-time analysis. This indicates that the unifying response-time analysis would benefit from the improved jitter term as well.

Although the experiments in this section showcase the potential improvement for the unifying response time analysis, a proper integration of the jitter term is part of future work. In case the unifying response time analysis can be proven by shifting the analysis to the right-hand side instead of to the left-hand side *in future work*, then we conjecture that the improved jitter-term $R_i - R_i^-$ presented in this work can be utilized directly in the unifying analysis as well, i.e., Equation (24) could be replaced by

$$R_k \geq C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{R_k + Q_i^{\vec{x}} + (1 - x_i)(R_i - R_i^-)}{T_i} \right\rceil C_i. \tag{25}$$

Therefore, if this conjecture holds, this new analysis will dominate all previous analytical approaches for sporadic tasks with dynamic self-suspension under uniprocessor fixed-priority scheduling. This is left as an open problem.

9 Conclusion

Jitter-based analysis is a common approach to bound the worst-case response time of self-suspending tasks. However, to date the seemingly trivial jitter term $J_i = R_i - C_i$ (that is, the worst-case response time minus the worst-case execution time of the higher-priority tasks) is the only safe result.

In this work we observe that this jitter term is pessimistic in the sense that this jitter can only occur for all higher-priority tasks in a system if carry-in workload of multiple tasks is executed simultaneously. Building upon that observation, we reduce the jitter term to $J_i = R_i - R_i^-$, which incorporates a lower bound $R_i^- \geq C_i$ that it takes to execute the carry-in workload. The new jitter-term requires a fundamentally different analytical approach where the analysis window is shifted *to the right-hand side* instead of to the left-hand side.

With the new jitter-term, our analysis dominates the previous jitter-based analysis. Moreover, we show that for synthesized tasksets an improvement of the worst-case response time over the previous jitter-based analysis can be found in up to 55.89% of the tasksets.

References

1 Federico Aromolo, Alessandro Biondi, and Geoffrey Nelissen. Response-time analysis for self-suspending tasks under EDF scheduling. In *34th Euromicro Conference on Real-Time Systems, ECRTS*, pages 13:1–13:18, 2022. `doi:10.4230/LIPIcs.ECRTS.2022.13`.

2 Neil C. Audsley and Konstantinos 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. `doi:10.1109/ECRTS.2004.12`.

3 Neil C. Audsley and Konstantinos Bletsas. Realistic analysis of limited parallel software / hardware implementations. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 388–395, 2004. `doi:10.1109/RTTAS.2004.1317285`.

4 Matthias Becker, Dakshina Dasari, and Daniel Casini. On the qnx ipc: Assessing predictability for local and distributed real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, 2023.

5 Konstantinos Bletsas and Neil 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. `doi:10.1109/RTCSA.2005.48`.

6 Konstantinos Bletsas, Neil C. Audsley, Wen-Hung Huang, Jian-Jia Chen, and Geoffrey Nelissen. Errata for three papers (2004-05) on fixed-priority scheduling with self-suspensions. *Leibniz Trans. Embed. Syst.*, 5(1):02:1–02:20, 2018. `doi:10.4230/LITES-v005-i001-a002`.

7 B.B. Brandenburg. Improved analysis and evaluation of real-time semaphore protocols for P-FP scheduling. In *RTAS*, 2013. `doi:10.1109/RTAS.2013.6531087`.

8 Bj"orn B. Brandenburg. *Multiprocessor Real-Time Locking Protocols*, pages 347–446. Springer Nature Singapore, Singapore, 2022. `doi:10.1007/978-981-287-251-7_10`.

9 Alan Burns and Gordon Baxter. *Time Bands in Systems Structure*, pages 74–88. Springer, January 2006.

10 A. Carminati, R.S. de Oliveira, and L.F. Friedrich. Exploring the design space of multiprocessor synchronization protocols for real-time systems. *Journal of Systems Architecture*, 60(3):258–270, 2014.

11 Daniel Casini, Alessandro Biondi, Geoffrey Nelissen, and Giorgio C. Buttazzo. Partitioned fixed-priority scheduling of parallel tasks without preemptions. In *2018 IEEE Real-Time Systems Symposium, (RTSS)*, pages 421–433, 2018. `doi:10.1109/RTSS.2018.00056`.

12 Daniel Casini, Paolo Pazzaglia, Alessandro Biondi, and Marco Di Natale. Optimized partitioning and priority assignment of real-time applications on heterogeneous platforms with hardware acceleration. *J. Syst. Archit.*, 124:102416, 2022. `doi:10.1016/j.sysarc.2022.102416`.

13 Jian-Jia Chen, Tobias Hahn, Ruben Hoeksma, Nicole Megow, and Georg von der Brüggen. Scheduling self-suspending tasks: New and old results. In Sophie Quinton, editor, *31st Euromicro Conference on Real-Time Systems, ECRTS*, volume 133 of *LIPIcs*, pages 16:1–16:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.ECRTS.2019.16`.

14 Jian-Jia Chen, Wen-Hung Huang, and Geoffrey Nelissen. A note on modeling self-suspending time as blocking time in real-time systems. *Computing Research Repository (CoRR)*, 2016. http://arxiv.org/abs/1602.07750.

15 Jian-Jia Chen and Cong Liu. Fixed-relative-deadline scheduling of hard real-time tasks with self-suspensions. In *Real-Time Systems Symposium (RTSS)*, pages 149–160, 2014. doi:10.1109/RTSS.2014.31.

16 Jian-Jia Chen, Geoffrey Nelissen, and Wen-Hung Huang. A unifying response time analysis framework for dynamic self-suspending tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 61–71, 2016. doi:10.1109/ECRTS.2016.31.

17 Jian-Jia Chen, Geoffrey Nelissen, Wen-Hung Huang, Maolin Yang, Björn B. Brandenburg, Konstantinos Bletsas, Cong Liu, Pascal Richard, Frédéric Ridouard, Neil C. Audsley, Raj Rajkumar, Dionisio de Niz, and Georg von der Brüggen. Many suspensions, many problems: a review of self-suspending tasks in real-time systems. *Real Time Syst.*, 55(1):144–207, 2019. doi:10.1007/s11241-018-9316-9.

18 Jian-Jia Chen, Georg von der Brüggen, Wen-Hung Huang, and Cong Liu. State of the art for scheduling and analyzing self-suspending sporadic real-time tasks. In *23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA*, pages 1–10, 2017. doi:10.1109/RTCSA.2017.8046321.

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

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

21 Zheng Dong, Cong Liu, Soroush Bateni, Kuan-Hsun Chen, Jian-Jia Chen, Georg von der Brüggen, and Junjie 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. doi:10.1109/RTAS.2018.00026.

22 David Griffin, Iain Bate, and Robert I. Davis. Generating utilization vectors for the systematic evaluation of schedulability tests. In *41st IEEE Real-Time Systems Symposium, RTSS 2020, Houston, TX, USA, December 1-4, 2020*, pages 76–88. IEEE, 2020. doi:10.1109/RTSS49844.2020.00018.

23 Mario Günzel and Jian-Jia Chen. Correspondence article: Counterexample for suspension-aware schedulability analysis of EDF scheduling. *Real Time Syst.*, 56(4):490–493, 2020.

24 Mario Günzel and Jian-Jia Chen. A note on slack enforcement mechanisms for self-suspending tasks. *Real Time Systems Journal*, 57(4):387–396, 2021. URL: https://link.springer.com/article/10.1007/s11241-020-09362-z.

25 Mario Günzel, Niklas Ueter, and Jian-Jia Chen. Suspension-aware fixed-priority schedulability test with arbitrary deadlines and arrival curves. In *42nd IEEE Real-Time Systems Symposium, RTSS*, pages 418–430, 2021. doi:10.1109/RTSS52674.2021.00045.

26 Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen. Suspension-aware earliest-deadline-first scheduling analysis. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 39(11):4205–4216, 2020. doi:10.1109/TCAD.2020.3013095.

27 Mario Günzel, Georg von der Brüggen, Kuan-Hsun Chen, and Jian-Jia Chen. EDF-like scheduling for self-suspending real-time tasks. In *IEEE Real-Time Systems Symposium, (RTSS)*, pages 172–184, 2022. doi:10.1109/RTSS55097.2022.00024.

28 Wen-Hung Huang and Jian-Jia 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.

29 Wen-Hung Huang, Jian-Jia Chen, Husheng Zhou, and Cong 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. doi:10.1145/2744769.2744891.

30 Wen-Hung Huang, Maolin Yang, and Jian-Jia Chen. Resource-oriented partitioned scheduling in multiprocessor systems: How to partition and how to share? In *Real-Time Systems Symposium (RTSS)*, pages 111–122, 2016.

31 M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, May 1986. doi:10.1093/comjnl/29.5.390.

32 In-Guk Kim, Kyung-Hee Choi, Seung-Kyu Park, Dong-Yoon Kim, and Man-Pyo Hong. Real-time scheduling of tasks that contain the external blocking intervals. In *RTCSA*, pages 54–59, 1995. doi:10.1109/RTCSA.1995.528751.

33 Junsung Kim, Björn Andersson, Dionisio de Niz, Jian-Jia Chen, Wen-Hung Huang, and Geoffrey Nelissen. Segment-fixed priority scheduling for self-suspending real-time tasks. Technical Report CMU/SEI-2016-TR-002, CMU/SEI, 2016.

34 Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real world automotive benchmarks for free. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

35 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.

36 Karthik Lakshmanan, Dionisio de Niz, and Ragunathan Rajkumar. Coordinated task scheduling, allocation and synchronization on multiprocessors. In *Real-Time Systems Symposium (RTSS)*, pages 469–478, 2009. doi:10.1109/RTSS.2009.51.

37 John P. Lehoczky, Lui 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. doi:10.1109/REAL.1989.63567.

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

39 Cong Liu and Jian-Jia Chen. Bursty-interference analysis techniques for analyzing complex real-time task models. In *Real-Time Systems Symposium (RTSS)*, pages 173–183, 2014.

40 Jane W. S. Liu. *Real-Time Systems*. Prentice Hall PTR, 1st edition, 2000.

41 Wei Liu, Jian-Jia Chen, Anas Toma, Tei-Wei Kuo, and Qingxu Deng. Computation offloading by using timing unreliable components in real-time systems. In *Design Automation Conference (DAC)*, volume 39:1 – 39:6, 2014. doi:10.1145/2593069.2593109.

42 Li Ming. Scheduling of the inter-dependent messages in real-time communication. In *Proc. of the First International Workshop on Real-Time Computing Systems and Applications*, 1994.

43 Geoffrey Nelissen, José Fonseca, Gurulingesh Raravi, and Vincent Nélis. Timing Analysis of Fixed Priority Self-Suspending Sporadic Tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 80–89, 2015.

44 Bo Peng and Nathan 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. doi:10.1109/RTCSA.2016.15.

45 R. Rajkumar. Dealing with Suspending Periodic Tasks. Technical report, IBM T. J. Watson Research Center, 1991. URL: http://www.cs.cmu.edu/afs/cs/project/rtmach/public/papers/period-enforcer.ps.

46 Ragunathan Rajkumar, Lui Sha, and John P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS '88)*, pages 259–269, 1988.

47 Lea Schönberger, Wen-Hung Huang, Georg von der Brüggen, Kuan-Hsun Chen, and Jian-Jia 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, 2018. doi:10.1109/RTCSA.2018.00027.

48 Ken Tindell, Alan Burns, and Andy J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real Time Syst.*, 6(2):133–151, 1994. doi:10.1007/BF01088593.

49 Niklas Ueter, Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen. Hard real-time stationary gang-scheduling. In *33rd Euromicro Conference on Real-Time Systems, ECRTS*, pages 10:1–10:19, 2021. doi:10.4230/LIPIcs.ECRTS.2021.10.

50 Georg von der Brüggen, Alan Burns, Jian-Jia Chen, Robert I. Davis, and Jan Reineke. On the trade-offs between generalization and specialization in real-time systems. In *28th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA*, pages 148–159, 2022. doi:10.1109/RTCSA55878.2022.00022.

51 Georg von der Brüggen, Wen-Hung Huang, and Jian-Jia Chen. Hybrid self-suspension models in real-time embedded systems. In *International Conference on Real-Time Computing Systems and Applications (RTCSA)*, 2017.

52 Beyazit Yalcinkaya, Mitra Nasri, and Björn B. Brandenburg. An exact schedulability test for non-preemptive self-suspending real-time tasks. In *Design, Automation & Test in Europe Conference & Exhibition, DATE*, pages 1228–1233, 2019. doi:10.23919/DATE.2019.8715111.