Generalized Weakly Hard Schedulability Analysis for Real-Time Periodic Tasks

Pazzaglia, P., Sun, Y., & Di Natale, M. (2020). Generalized Weakly Hard Schedulability Analysis for Real-Time Periodic Tasks. *ACM Transactions on Embedded Computing Systems*, *20*(1), Article 3. https://doi.org/10.1145/3404888

Published in:
ACM Transactions on Embedded Computing Systems

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Generalized Weakly Hard Schedulability Analysis for Real-Time Periodic Tasks

PAOLO PAZZAGLIA*, Scuola Superiore Sant'Anna, Pisa, Italy
YOUCHENG SUN, Queen's University Belfast, UK
MARCO DI NATALE, Scuola Superiore Sant'Anna, Pisa, Italy

The weakly hard real-time model is an abstraction for applications, including control systems, that can tolerate occasional deadline misses, but can also be compromised if a sufficiently high number of late terminations occur in a given time window. The weakly hard model allows to constrain the maximum number of acceptable missed deadlines in any set of consecutive task executions. A big challenge for weakly hard systems is to provide a *schedulability analysis* that applies to a general task model, while avoiding excessive pessimism. In this work, we develop a general weakly hard analysis based on a Mixed Integer Linear Programming (MILP) formulation. The analysis applies to constrained-deadline periodic real-time systems scheduled with fixed priority and no knowledge of the task activation offsets, while allowing for activation jitter. Our analysis considers two common policies for handling missed deadlines, i.e. (i) letting the job continue until completion or (ii) killing its execution immediately. For this policy, ours is the first and only m-k analysis currently available. Experiments conducted on randomly generated task sets show the applicability and accuracy of the proposed technique as well as the improvements with respect to competing techniques.

CCS Concepts: • **Computer systems organization → Real-time system specification**; **Embedded software**.

Additional Key Words and Phrases: Weakly hard real-time systems, schedulability analysis, periodic real-time tasks, activation jitter, deadline-miss handling strategies.

1 INTRODUCTION

The concept of real-time task abstracts the behavior of a software program, characterized by an activation pattern and a *deadline* constraint on its execution. In *hard* real-time scheduling, all jobs are required to complete before their deadlines, and a proper analysis is built to check this property for the worst case condition. Schedulability analysis for hard real-time systems has been investigated in the past and solutions exist for many cases of interest. However, fewer results exist for task sets that may tolerate a limited number of deadline misses. Filling this gap is of great interest because this behavior is indeed representative of a large number of real-world applications, where (few) late terminations do not significantly harm the application, including control systems [26]. Intuitively, each missed deadline will move the system state closer to a faulty condition, while a sufficient number of timely completions will restore a safe working condition for the system. Considering these systems as hard real-time would result in needless pessimism and possibly over-provisioning of resources.

The *weakly hard* scheduling model [7, 25] gained interest as a way to describe systems that are robust to (some) deadline violations. This requirement is usually defined using one or more pairs (m, K) representing the maximum number m of missed deadlines that can be tolerated every K activations (also known as the m-K model). Analysis methods for hard-type systems do not apply to weakly hard systems: this is mostly because the critical instant theorem does not hold anymore, and therefore the identification of the worst case scenario on which to formulate the analysis becomes much more difficult. Most available results apply to fixed-offset systems, i.e. when the

*Paolo Pazzaglia is now with Universität des Saarlandes, Saarbücken, Germany

Authors' addresses: Paolo Pazzaglia, Scuola Superiore Sant'Anna, Pisa, Italy, p.pazzaglia@sssup.it; Youcheng Sun, Queen's University Belfast, UK, youcheng.sun@qub.ac.uk; Marco Di Natale, Scuola Superiore Sant'Anna, Pisa, Italy, marco@sssup.it.

initial offset of all tasks is known. However, this hypothesis can be a serious limitation, since initial offsets are not always defined, or even possible to enforce. In addition, the analysis of fixed-offset systems is very sensitive to time errors and drifts that may occur in the task activation pattern, which are unfortunately difficult to prevent in real systems.

Motivated by these challenges, this paper develops a generalized weakly-hard analysis for offset-free systems that applies to periodic tasks scheduled by fixed priority on a uniprocessor, with a possible release jitter. The formulation is based on an MILP encoding, where the beginning of the busy period is represented as a problem variable. Two common strategies for handling the deadline miss event are considered: *continue* the job execution until completion, or immediately *kill* it.

The weakly hard analysis developed with our MILP model checks if a given constraint (m, K) is at least an upper bound of the weakly hard properties of the task. In other words, the outcome of the analysis guarantees that the target task will have no more than m deadline misses out of any K successive jobs. The efficiency of this analysis benefits from the relaxation from integers to real-valued variables for counting the number of higher priority interfering jobs, with binary variables used in an additional set of constraints to restore accuracy. Two formulations are presented for each strategy. The first one leverages a limited set of variables and constraints, guaranteeing a faster runtime but providing a bound on deadline misses which is possibly pessimistic. An advanced (but computationally heavier) formulation is then proposed for obtaining a tighter bound on (m, K) constraints, by checking *schedulability* properties at some selected points in time.

A preliminary version of the analysis in this paper has been presented at EMSOFT'17 [37]. The major extensions developed in the present work are briefly summarized as follows.

- In section 5.2 we propose a refined analysis for the *job-continue* strategy (extending the pessimistic one presented in [37]), which leverages additional constraints to provide a tighter estimate of the maximum number of deadline misses.
- In Section 6 we provide an analysis for the *job-kill strategy*, i.e. when a job that misses the deadline is forcibly terminated. This analysis is an original contribution w.r.t. [37] (and other prominent works like [17, 38]) that adopts a *continue* strategy. A refined formulation for this strategy (but more computationally expensive) is provided in Section 6.2. To our knowledge, this is the first analysis for this deadline management policy to ever appear in the literature.
- In Section 7, we compare the two deadline miss handling polices, and we show the improvements on the deadline miss estimate when using the refined analysis (on average an increase in the schedulability ratio of nearly 30% compared with the original approach in [37]).

2 RELATED WORK

Since the seminal work of Liu and Layland [24], the *hard* real-time model, where every job is required to complete before its deadline, has been the subject of intensive study from the research community. According to this model, missing a deadline is a potentially disruptive event that must be always avoided. The reasons behind the disproportionate interest on the part of researchers in hard analysis techniques probably lie in the simplicity of the model – both to be understood and analyzed – and the seemingly natural fit to real-world systems, in particular safety-critical systems. Its popularity also benefits from the existence of the critical instant, i.e., the condition where all tasks are synchronously activated, which leads to the worst-case conditions of the response time for every task in the system. As a result, in hard real-time systems it is sufficient to investigate the specific pattern of task activations originating from the critical instant [12],[33]. In the case of periodic tasks with explicit initial offsets, Leung and Whitehead [23] proved that it is necessary and sufficient to simulate the system using the worst-case execution time for all tasks, starting from any job activation, in a bounded time interval, since the schedule of periodic tasks will repeat

itself. However, Baruah and Burns showed in [6] that the result of this test is very sensitive to small variations in the task parameters, including the initial offset.

While hard deadline requirements may be necessary to a restricted number of safety-critical systems, there are many more, including control systems, that can tolerate (a limited number of) timing violations, with some research works showing how several systems can actually benefit from oversampling, even when this comes at the price of some deadline miss [26, 28]. In these cases, the hard schedulability analysis may be too pessimistic. The so-called *weakly* hard real-time schedulability analysis [7] was introduced for such systems, to address the problem of bounding the *maximum number of missed deadlines* that may happen in a given number of activations of a task. The fondant constraint of this analysis is defined by the pair (m, K), associated to a task, which means that no more than m deadline misses must occur for any K consecutive activations of that task. The (m, K) constraints are first used in [25], where the authors proposed a dynamic priority assignment for streams with weakly-hard requirements to reduce the probability of (m, K) violations. Anyway, the analysis presented in [7] and in many other works is based on the assumption that the initial offset of each task in the system is known and defined. The weakly-hard analysis is performed starting from any activation instant of one of the periodic tasks and repeated for each other activation until a large enough time interval is covered, checking that the number of deadline misses do not exceed the (m, K) constraints. While the assumption of periodic tasks is quite reasonable when considering real applications, the requirement of knowing all activation offsets may be too strict and potentially undermine the robustness of the analysis: in fact, even if the weakly hard schedulability is guaranteed for given initial offsets, it may unexpectedly fail when some of these offsets are slightly changed.

A different approach, presented in [30], consists in building a *worst-case analysis* by describing the system as the superposition of activations due to typical behavior, and sporadic (rare) activations due to overload conditions. This sporadic behavior is obtained by studying those rare events that may happen in the behavior of the system at runtime. Building over such an assumption, [17] and [38] proposed a two-steps method for weakly hard analysis, checking that (i) the system is schedulable under the typical behavior (using classical hard analysis), and (ii) in case of overload conditions, the given (m, k) constraint is always guaranteed. Authors in [20] follow a similar approach, using real-time calculus to analyze the worst-case busy period (in duration and lateness) when a temporary overload occurs because of an error in the assumptions on the task worst-case execution time. The biggest limit of these methods is that they require the definition of a task model that is possibly artificial, since they assume that the causes of possible overload are identifiable and known. All works discussed so far assume that a job continues executing even after it misses the deadline. By contrast, the weakly hard analysis in this paper also considers the immediate termination model to handle deadline misses and includes a comparison between the two approaches. Terminating a task upon a timing fault (and possibly enabling some fault management routine) is not only an interesting approach on the theoretical side, but is also a quite common fault management strategy in many practical systems.

The analysis of overload conditions is also closely related to the control-scheduling co-design problem [5]. When a controller is implemented in a real-time system, studying how its response time behaves in overload conditions is of particular importance since it may impact the overall performance of the controlled system [39, 40]. In control-scheduling co-design problems, the (m, K) model has generated great interest [14, 31] and has been used in [10, 11, 35, 36] as a constraint on the maximum number of jobs that can be dropped in a given time window, where those values are determined based on the minimum required level of quality of control. An integrated approach for controller synthesis is presented in [4], by finding task parameters that guarantee that the system is stable and provides the required control performance. This idea has been further extended in

[3], moving from uniprocessors to distributed cyber physical systems. In [16] a similar analysis using communication via FlexRay is considered. Other works [13, 15, 29] studied the impact of missed deadlines on control performance considering different strategies in case of deadline misses. In [18], the problem of verifying the safety of a system with weakly hard bounds is addressed. A control design which explicitly takes into account missed deadlines in a probabilistic fashion is recently explored in [28]. Several recent papers like [1, 2, 34] elaborate the use of weakly hard model in specific applications, including energy variable systems, finite queue platforms, cost-aware scheduling and networked environments, showing a constantly growing interest in the topic.

3 SYSTEM MODEL

In this work, a set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ of n real-time periodic tasks τ is scheduled by fixed priority with preemption on a uniprocessor platform. Tasks in $\mathcal{T}$ are indexed by priority, from higher to lower (i.e., τ_j has higher priority than τ_i if $j < i$). Each task τ_i is defined by the tuple $(C_i, D_i, T_i, \mathcal{J}_i)$, where C_i is its worst-case execution time (WCET), T_i is the activation period, D_i is the relative deadline (with $C_i \leq D_i \leq T_i$) and $\mathcal{J}_i$ is the (possible) maximum release jitter, with the constraint that $\mathcal{J}_i + C_i \leq D_i$. The total utilization of the task set is defined as $U = \sum_{i=1}^{n} U_i$, where $U_i = C_i/T_i$.

Each instance of τ_i is a *job* defined as $J_{i,k}$, where $k = 1, 2, \ldots$ represents the job index. Each job $J_{i,k}$ is defined by its arrival time $a_{i,k}$, finish time $f_{i,k}$, and absolute deadline $d_{i,k}$. When $\mathcal{J}_i = 0$, $a_{i,k+1} - a_{i,k} = T_i$ for any two successive jobs τ_i. For $\mathcal{J}_i \geq 0$, the distance between two successive activations is in the range $[T_i - \mathcal{J}_i, T_i + \mathcal{J}_i]$. Moreover, the starting offset of each task τ_i is not known and so is the value of the first job activation time $a_{i,1}$. A job $J_{i,k}$ is schedulable if it finishes its execution before its deadline ($f_{i,k} \leq d_{i,k}$), otherwise it is not schedulable, and *misses* its deadline.

The elapsed time between a job activation and finish time (i.e., $f_{i,k} - a_{i,k}$) is called *response time*. The worst-case response time (WCRT), is computed as $R_i = \max_k \{f_{i,k} - a_{i,k}\}$. Similarly, the best-case response time (BCRT) is defined as $r_i = \min_k \{f_{i,k} - a_{i,k}\}$. A task τ_i is then deemed schedulable if and only if $R_i \leq D_i$. If $R_i > D_i$ the task is non-schedulable. Note that $\mathcal{T}$ may contain multiple non-schedulable tasks. To prevent trivial cases, we assume hereafter that $r_i \leq D_i$, $\forall \tau_i$. Moreover, the assumption $U < 1$ is enforced, meaning that each job is guaranteed to complete its requested execution at some point in time, even if it misses its deadline. The BCRT r_i and the WCRT R_i of each task are computed in advance using established techniques such as those in [9, 22, 32].

A job $J_{i,k}$ is *active* at time t if it has not completed its current execution, i.e. if $a_{i,k} \leq t < f_{i,k}$. A *level-i busy period* is then defined as a time interval where, at each point in time, at least one job with priority greater than or equal to τ_i is active. As an example, in Figure 1, both $[s_0, f_2)$ and $[a_3, f_3)$ are level-3 busy periods. Note that the definition of busy period given above is indeed a generalization of the maximal level-i busy period defined in [22]. Similarly, we introduce the definition of *level-i processor idle time* as a time interval where the processor is not occupied by τ_i or any other higher priority task. Finally, since fixed priority scheduling analysis is *sustainable* [6], if a job is not schedulable, it will not become schedulable by increasing its execution time. For this reason, we simply assume that a task requests its WCET every time it is activated.

3.1 The Weakly Hard Model

Consider a task $\tau_i \in \mathcal{T}$, with $i > 1$, that has $R_i > D_i$. The weakly hard schedulability properties of τ_i are checked by finding an upper bound on the number of missed deadlines experienced by the task in the worst case. The outcome of the analysis is presented as a set of pairs (m, K), where m represents the maximum number of missed deadlines of τ_i considering every possible sequence of K successive activations.

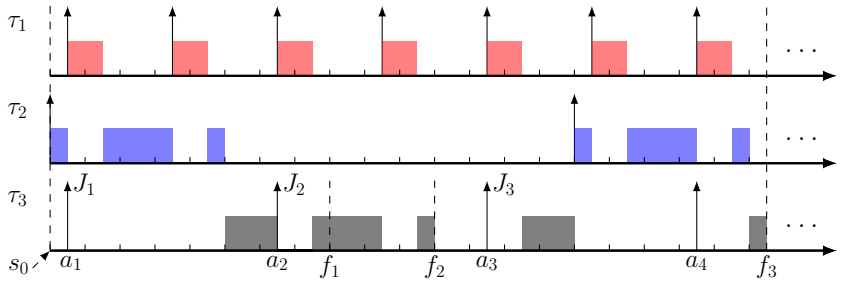

Fig. 1. A problem window with 3 job windows

3.1.1 Strategies for handling missed deadlines. Multiple choices can be made to manage a missed deadline. In this work, we consider two common approaches: the *job-continue* and the *job-kill* policies. With the job-continue policy, jobs are always executed until their completion. This approach minimizes the waste of processor time, but *self-pushing* effects may occur when having multiple pending jobs at the same time, thus potentially increasing the response time of the following instances. The other strategy considered in this work is *killing* the job when a deadline is missed, i.e., the computation is discarded at the deadline instant if the job has not completed. A schedulability analysis is presented for both strategies, allowing to compare the results.

For ease of notation, the k-th job of τ_i will be denoted as J_k (without the task index) when the context is clear. Similarly, given a job J_k, its activation time and finish time are denoted as a_k and f_k, respectively. The interval $[a_k, a_{k+1})$ is the *k-th job window* of τ_i. The *problem window* under analysis is $[s_0, f_K)$, where s_0 is the earliest instant of the level-i busy window for the first activation a_1 of τ_i. Without loss of generality, we have $s_0 = 0$.

As an example, consider a system of 3 tasks with no jitter: $(C_1 = 1, D_1 = T_1 = 3)$, $(C_2 = 3, D_2 = T_2 = 15)$ and $(C_3 = 2, D_3 = T_3 = 6)$, where $K = 3$ and τ_3 is the target task. Figure 1 shows a scenario where 2 (J_1 and J_3) out of 3 jobs in the problem window $[s_0, f_3)$ miss the deadline, with $a_1 = 0.5$. If the problem window starts with all tasks synchronously activated in s_0, only J_1 misses its deadline.

4 THE SOLUTION MODEL

4.1 Worst-Case Schedule

A *worst-case schedule* of τ_i, defined as $S_w(K)$, is a problem (time) window of K consecutive jobs of τ_i, $\{J_1, J_2, \ldots, J_K\}$, such that the *maximum number of them experience a deadline miss*. Different schedules of K consecutive jobs can produce the same worst-case number of deadline misses, but, from the schedulability analysis point of view, it is sufficient to find any one of them. In order to reduce the problem space, we identify some properties that are necessary for those problem windows that maximize the number of deadline misses for τ_i.

Lemma 1. *Given K consecutive jobs of τ_i, there exists at least one worst-case schedule $S_w(K) = \{J_1, J_2, \ldots, J_K\}$ such that both the following conditions hold:*

(1) *the job ending before the beginning of the problem window (namely J_0) is* schedulable*; and*
(2) *the first job of the problem window (J_1) is* non-schedulable.

Proof. The proof is by contradiction. We demonstrate that if there exists a worst-case schedule that violates either condition (1) or (2) then there must exist at least one additional schedule with the same number of misses (therefore also worst-case) that satisfies both of them. The existence of such schedule is proven by construction. Because of the hypotheses that $R_i > D_i$ and $r_i \leq D_i$, there must be at least one schedule of τ_i where a schedulable job is followed by a non-schedulable one.

Type	Variables	Annotations
$\mathbb{R}$	a_k	Activation time of J_k.
	L_k	Segment of busy execution of higher priority tasks before a_k, when $f_{k-1} \leq a_k$.
	α_j	Offset of the first job activation of τ_j w.r.t. s_0.
	ι_k	Level-i idle time within the k-th job window.
$\mathbb{B}$	b_k	$b_k = 0$ if J_k is schedulable; otherwise $b_k = 1$
	b_j^o	$b_j^o = 0$ if $\alpha_j = 0$; otherwise $b_j^o = 1$.
	β_k	$\beta_k = 0$ if $f_k \leq a_{k+1}$; otherwise $\beta_k = 1$.

Table 1. MILP variables that are common to both strategies.

Assume there is a worst-case schedule $S'_w(K) = \{J_x, \ldots, J_{x+K-1}\}$ such that condition (1) of the Lemma does not hold for $S'_w(K)$, i.e., J_{x-1} is *not schedulable*. By shifting the problem window one job to the left, we obtain a sequence of K jobs $S_L(K) = \{J_{x-1}, \ldots, J_{x+K-2}\}$ that has *at least as many misses* as $S'_w(K)$. If $S'_w(K)$ is a worst-case schedule, then $S_L(K)$ has the same number of misses (cannot be higher) and must also be a worst-case schedule. However, by construction, $S_L(K)$ satisfies condition (2) of the Lemma (its first job J_{x-1} is non-schedulable). If the last job before the beginning of the K-job window of $S_L(K)$ is not schedulable ($S_L(K)$ does not satisfy condition (1)), then we can shift again the problem window by one job, obtaining another worst-case schedule of K jobs that satisfies condition (2). Eventually (by iteration), we find a sequence of jobs $\{J_{x-r}, \ldots, J_{x+K-r-1}\}$ that is a worst-case schedule satisfying condition (2), for which the last job before the beginning of the window J_{x-r-1} is schedulable, thus satisfying also condition (1).

Assume condition (2) of the Lemma does not hold for $S'_w(K)$, i.e., J_x is *schedulable*. Then, the sequence $S_R(K) = \{J_{x+1}, \ldots, J_{x+K}\}$ has *at least as many deadline misses* as $S'_w(K)$. Since $S'_w(K)$ is a worst-case schedule, then also $S_R(K)$ must be a worst-case schedule. Additionally, $S_R(K)$ satisfies condition (1) of the Lemma (since the last job before the time window J_x is schedulable). If J_{x+1} is schedulable and $S_R(K)$ does not satisfy condition (2), we can shift the window one more job to the right, finding another worst-case schedule that still satisfies condition (1). Eventually, we find a sequence $\{J_{x+q}, \ldots, J_{x+K+q-1}\}$ that is a worst-case schedule where J_{x+q} is not schedulable (condition (2)), and J_{x+q-1} is schedulable (condition (1)). Thus, the Lemma follows. □

In the following, without loss of generality, we will consider sequences $\{J_1, \ldots, J_K\}$ that satisfy the lemma presented above (i.e., having J_0 schedulable and J_1 not schedulable).

4.2 Common Variables and Basic MILP Constraints

An MILP formulation requires constraints defined as linear inequalities (an equality $a = b$ can be obtained using $a \leq b$ and $b \geq a$), and a linear objective function. We use variables that can assume *positive real values* (including zero), labeled by $\mathbb{R}$, or *Boolean variables*, labeled by $\mathbb{B}$, that may assume only $\{0, 1\}$. M is a sufficiently big constant value used to encode conditional constraints (a standard technique known as big-M). In the following, we first present the constraints that are common to both the job-continue and job-kill strategies. For sake of brevity, proofs are omitted when trivial. A summary of the variables used in this section can be found in Table 1. Those constraints that are specific to each policy will be presented next, in dedicated sections.

4.2.1 Starting instant of the level-i busy windows. Each job J_k of τ_i activated at time a_k can be interfered by pending executions of higher priority tasks that are activated before a_k but are not yet completed at a_k. Let $a_k - L_k$ be the start of the level-i busy period that includes a_k: the variable $L_k \in \mathbb{R}$ represents the portion of the level-i busy period for job J_k that extends to the earliest such activation, when $f_{k-1} \leq a_k$. A trivial constraint on the size of the busy windows is the following.

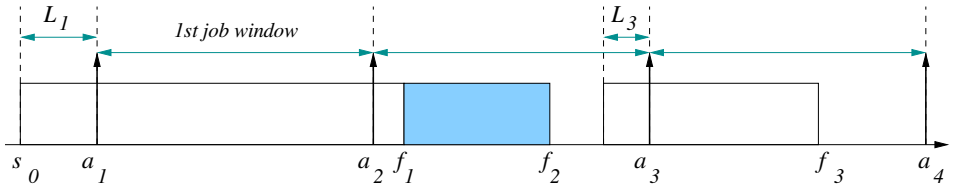

Fig. 2. Notation for the definition of a problem window

Constraint 1. *The size of the interval L_k can be bounded as:*

$$\forall k, \quad 0 \leq L_k \leq T_i + \mathcal{J}_i - r_i. \tag{1}$$

If the busy period that includes a_k starts earlier than a_{k-1}, it is already accounted for by the definition of L_{k-1}. Hence, L_k is well-defined only when $f_{k-1} \leq a_k$. An illustrative example of the formulation of L_k is in Figure 2: here, since $f_1 > a_2$, L_2 is not defined. In the following, additional checks are provided to those constraints that involve L_k, to cope with the case in which $f_{k-1} > a_k$. According to condition (1) of Lemma 1 (which translates into $f_0 \leq a_1$), L_1 is always well-defined. Thus, the start time of the problem window is $s_0 = a_1 - L_1 = 0$.

4.2.2 Activation instants and jitter. An MILP variable $a_k \in \mathbb{R}$ is introduced to represent the activation instant of J_k. If $\mathcal{J}_i = 0$, the activation time of J_k is a simple function of L_1, i.e., $a_k = L_1 + (k-1)T_i$ (L_1 is the initial offset of τ_i with respect to s_0). In the general case, when tasks are possibly released with jitter, the distance between the activation times of two jobs is not fixed. The following constraint then trivially holds from the definition.

Constraint 2. *The value of the activation instant a_k of job J_k is bounded by:*

$$\forall k, \quad L_1 + (k-1) \cdot T_i \leq a_k \leq L_1 + (k-1) \cdot T_i + \mathcal{J}_i. \tag{2}$$

4.2.3 Offsets. The activation instant of the first job of a generic higher priority task τ_j in the problem window is defined by using a offset value $\alpha_j \in \mathbb{R}$ with respect to the initial start time s_0.

Constraint 3. *The values of the offsets for the interfering tasks are bounded by:*

$$\forall j < i, \quad 0 \leq \alpha_j \leq T_j - r_j + \mathcal{J}_j. \tag{3}$$

Proof. The upper bound of the constraint is proven by contradiction. Assume that the first job $J_{j,1}$ of an interfering task τ_j arrives at time $s_0 + T_j - r_j + \mathcal{J}_j + \epsilon$ with $\epsilon > 0$. This implies that the earliest instant where the previous job $J_{j,0}$ is activated is $s_0 - r_j + \epsilon$. Since any job of τ_j needs at least r_j time to finish, $J_{j,0}$ will be still active at s_0, which contradicts the hypothesis that s_0 is the earliest time instant such that from s_0 to a_1 the processor is fully occupied by higher priority tasks. Hence, the upper bound follows. The lower bound trivially comes from the definition of s_0. □

In order for s_0 to be the starting point of a busy period, at least one offset α_j (or L_1 for τ_i) must be equal to zero. Introducing a Boolean variable $b_j^o \in \mathbb{B}$ for each task τ_j with $j \leq i$, this constraint is enforced using the big-M encoding as follows.

Constraint 4. *The offsets are bounded as follows:*

$$\begin{aligned} &\forall j < i, \quad 0 \leq \alpha_j \leq b_j^o \cdot M \\ &\text{for } \tau_i, \quad 0 \leq L_1 \leq b_i^o \cdot M, \end{aligned} \tag{4}$$

such that also the upper bound $\sum_{j=1}^{i} b_j^o \leq i - 1$ holds.

The constraint enforces that at most $(i-1)$ tasks can be activated after the start point s_0. This means that at least one variable b_j^o is zero, thus from (4) at least one offset is equal to zero.

4.2.4 Finish time f_k. The variable $f_k \in \mathbb{R}$ represents the finish time of job J_k. For each job J_k, the time interval $[0, f_k)$ consists of multiple busy periods and processor idle times. The exact computation of the finish time of a job J_k requires finding the minimum fixed point (solution) of the recursive equation [19]:

$$\min\Big\{ f_k \mid \sum_{j<i} \left\lceil \frac{f_k - \alpha_j}{T_j} \right\rceil C_j + k \cdot C_i + Idle = f_k \Big\}. \tag{5}$$

This formulation cannot be easily encoded as an MILP constraint, because it would require a minimization function *for each activation k* of the task τ_i. A possible approach is to ignore the minimization term and accept any solution of the fixed-point equation, thus possibly including values larger than the correct one. This is computationally efficient, but clearly pessimistic. Additional constraints could be introduced to reduce the search space and remove (at least some of) the incorrect solutions. The precision of the computed solution is improved at the expense of additional execution time. The MILP encoding of the finish time equations for both approaches depends on the deadline management policy, and will be presented in detail in the related sections.

4.2.5 Deadline misses. A Boolean variable $b_k \in \mathbb{B}$ is introduced to indicate whether the job J_k in the problem window misses its deadline ($b_k = 1$) or not ($b_k = 0$). From condition (2) of Lemma 1, J_1 always misses its deadline.

Constraint 5. *Condition (2) of Lemma 1 is enforced by ensuring that $b_1 = 1$.*

4.2.6 Interference from pending jobs. In order to check whether a job J_k of τ_i interferes with the execution of the next job J_{k+1} , i.e. when $f_k > a_{k+1}$, a Boolean variable β_k is introduced as follows:

$$\beta_k = \begin{cases} 0 & \text{if } f_k \le a_{k+1}; \\ 1 & \text{otherwise.} \end{cases}$$

This can be re-written as an MILP constraint with a linear formulation.

Constraint 6. *The variable β_k representing the interference of the pending jobs of τ_i is bounded by:*

$$\forall k \quad -M \cdot \beta_k \le a_{k+1} - f_k < M \cdot (1 - \beta_k). \tag{6}$$

4.2.7 Total level-i idle time in a job window. In an arbitrary job window $[a_k, a_{k+1})$ of J_k, the processor may experience multiple level-i idle time intervals. The variable $\iota_k \in \mathbb{R}$ is introduced, to represent the sum of the level-i idle times (i.e. the total amount of processor idle time) in the k-th job window. A rough bound for ι_k follows.

Constraint 7. *The total level-i idle time ι_k in a generic window $[a_k, a_{k+1})$ is bounded by:*

$$\forall k \quad 0 \le \iota_k \le T_i + \mathcal{J}_i - r_i. \tag{7}$$

Proof. The upper-bound of ι_k represents the case in which the processor is occupied for the minimum amount of time by τ_i alone finishing with its BCRT r_i with maximum distance between two consecutive activations (i.e., $T_i + \mathcal{J}_i$). The lower bound is trivially zero. □

Note that if J_k finishes executing after a_{k+1}, then the processor is never idle inside the job window $[a_k, a_{k+1})$. This can be stated as an MILP constraint as follows.

Constraint 8. *The total level-i idle time ι_k in the window $[a_k, a_{k+1})$ is zero if $\beta_k = 1$, i.e.,:*

$$\forall k, \quad \iota_k \le M \cdot (1 - \beta_k). \tag{8}$$

4.2.8 Number of interfering jobs from higher priority tasks. When modeling a schedulability problem in MILP, the major complexity comes from computing the interference from higher priority tasks. A common approach is to count the number of jobs from each higher priority task that interfere with the execution of the task under analysis.

Given a job J_k of τ_i and a higher priority task τ_j, the number of jobs of τ_j within the time interval $[0, f_k)$ is defined as $If_{j,k}$. In case of no jitter for τ_j ($\mathcal{J}_j = 0$), this value can be computed as $If_{j,k} = \lceil \frac{f_k - \alpha_j}{T_j} \rceil$. In the general case (with jitter), it is a value constrained in the interval $\lceil \frac{f_k - \alpha_j - \mathcal{J}_j}{T_j} \rceil \leq If_{j,k} \leq \lceil \frac{f_k - \alpha_j + \mathcal{J}_j}{T_j} \rceil$. This formulation using ceiling terms is well-established, but in an MILP problem it requires to be rewritten using only linear constraints [22], as follows.

Constraint 9. *The number $If_{j,k}$ of jobs of τ_j activated in the window $[0, f_k)$ is bounded by:*

$$\forall j < i, \forall k, \quad \frac{f_k - \alpha_j - \mathcal{J}_j}{T_j} \leq If_{j,k} < \frac{f_k - \alpha_j + \mathcal{J}_j}{T_j} + 1, \tag{9}$$

In a similar way, we define $IL_{j,k}$ as the number of jobs of τ_j within the time interval $[0, a_k - L_k)$. Note that this value is meaningful only when $\beta_{k-1} = 0$. In this case, considering a generic jitter $\mathcal{J}_j$, $IL_{j,k}$ is constrained by $\lceil \frac{a_k - L_k - \alpha_j - \mathcal{J}_j}{T_j} \rceil \leq IL_{j,k} \leq \lceil \frac{a_k - L_k - \alpha_j + \mathcal{J}_j}{T_j} \rceil$. Again, this formulation can be written using linear inequalities as follows.

Constraint 10. *The number $IL_{j,k}$ of jobs of τ_j activated in the window $[0, a_k - L_k)$ (which is meaningful only if $\beta_{k-1} = 0$) is bounded by:*

$$\forall j < i, \forall k, \quad \frac{a_k - L_k - \alpha_j - \mathcal{J}_j}{T_j} - M \cdot \beta_{k-1} \leq IL_{j,k} < \frac{a_k - L_k - \alpha_j + \mathcal{J}_j}{T_j} + 1 + M \cdot \beta_{k-1}. \tag{10}$$

Note that when $\beta_{k-1} = 1$, Constraint 10 is trivially always verified. A numerical example for both $If_{j,k}$ and $IL_{j,k}$, based on the system of Figure 1, is presented in Table 2.

By definition, the variables $If_{j,k}$ and $IL_{j,k}$ count the number of jobs in window $[0, f_k)$ and $[0, a_k - L_k)$, respectively. For this reason, they should only assume integer values and appear as integer variables in the MILP formulation. However, this can be computationally expensive. To increase the computational efficiency, we can relax $If_{j,k}$ and $IL_{j,k}$ to real values, i.e. $If_{j,k} \in \mathbb{R}$ and $IL_{j,k} \in \mathbb{R}$. In this case, their correct (integer) values must then be restored by introducing additional constraints, as presented in the next Section.

4.2.9 Refining the interference from higher priority tasks. When $If_{j,k}$ and $IL_{j,k}$ are coded as real variables, we need to refine the constraints to let them assume only integer values. We obtain this by introducing two arrays of Boolean variables as follows.

Firstly, for each job J_k of τ_i consider the *k-th level-i busy window*, i.e. the time interval defined as **(i)** $[a_k - L_k, f_k)$ if $\beta_{k-1} = 0$, or **(ii)** $[f_{k-1}, f_k)$ if $\beta_{k-1} = 1$. An example of such intervals is in Figure 2, where the white boxes are level-i busy windows as defined for the case (i), while the blue box identifies one interval for the case (ii). For every such time interval and for each higher priority task τ_j, an array $\Gamma f_{j,k}$ of boolean variables $\Gamma f_{j,k}[p] \in \mathbb{B}$, is introduced. The index p is relative to the (higher priority) job activated in the time interval under analysis, i.e. $p = 1$ corresponds to the first job activated in the interval and so on. The value $\Gamma f_{j,k}[p] = 1$ (active) indicates that the p-th job of τ_j in the time interval interferes with J_k; otherwise, $\Gamma f_{j,k}[p] = 0$ (inactive, with no interference).

In order to be as general as possible, the number of jobs considered in the array $\Gamma f_{j,k}$ must be sized by considering the largest possible level-i busy window. A rough upper bound for the size of the array $\Gamma f_{j,k}$ (i.e. the maximum number of job activations of τ_j in the interval) is $\lceil \frac{R_i + T_i + \mathcal{J}_j - r_i}{T_j} \rceil$. Nevertheless, the number of jobs effectively contributing to the busy window of J_k may be lower

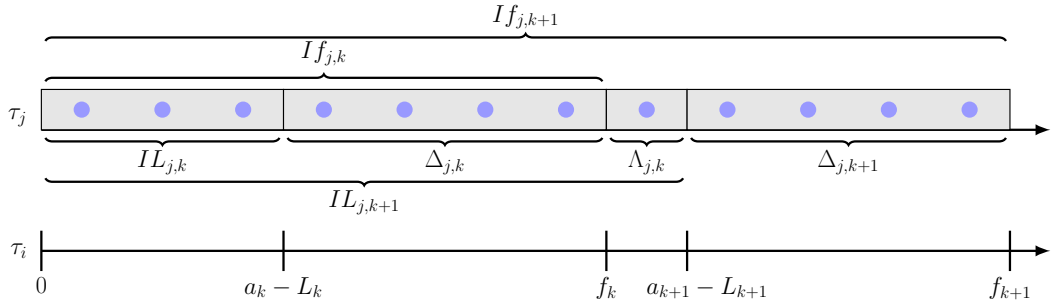

Fig. 3. Graphical representation of the interfering jobs of τ_j for different time windows. Each dot represents an activation of τ_j, while the variables in brackets count the number of jobs in the windows.

than this upper bound. As a general rule, if one job considered in the array $\Gamma f_{j,k}$ does not interfere with J_k, then no later job can interfere. This can be coded as follows.

Constraint 11. *If the p-th element of the array $\Gamma f_{j,k}$ corresponds to a job of τ_j that does not interfere with J_k, then no later job can interfere with J_k, i.e.,*

$$\forall j < i, \forall k, \quad \Gamma f_{j,k}[p] \geq \Gamma f_{j,k}[p+1]. \tag{11}$$

Proof. When $\Gamma f_{j,k}[p] = 1$, the next element can be either 0 or 1 without violating the constraint. If $\Gamma f_{j,k}[p] = 0$, the inequality (11) forces $\Gamma f_{j,k}[p+1] = 0$. □

Now, $\Gamma f_{j,k}$ can be leveraged to compute the number of jobs of τ_j interfering with the execution of J_k in the k-th level-i busy window. We introduce an auxiliary variable $\Delta_{j,k}$ defined as $\Delta_{j,k} := \sum_p \Gamma f_{j,k}[p]$, counting the number of jobs that interfere with τ_i in the k-th level-i busy window. Since $\Delta_{j,k}$ is a sum of Booleans (i.e., of variables that are either 0 or 1), it assumes only integer values. When $\beta_{k-1} = 0$, $\Delta_{j,k} = If_{j,k} - IL_{j,k}$, otherwise ($\beta_{k-1} = 1$) $\Delta_{j,k} = If_{j,k} - If_{j,k-1}$. To have a consistent formulation, we enforce $IL_{j,k} = If_{j,k-1}$ when $\beta_{k-1} = 1$ with the following constraint.

Constraint 12. *The equivalence $IL_{j,k} = If_{j,k-1}$ is enforced for the case $\beta_{k-1} = 1$ by:*

$$\forall j < i, \forall k, \quad -M \cdot (1 - \beta_{k-1}) \leq IL_{j,k} - If_{j,k-1} \leq M \cdot (1 - \beta_{k-1}). \tag{12}$$

Note that Constraint 12 is active when Constraint 10 is not active, and viceversa. Then, the following constraint holds.

Constraint 13. *Considering the number of jobs of τ_j activated in the interval $[0, f_k)$, it holds that:*

$$\forall j < i, \forall k, \quad IL_{j,k} + \Delta_{j,k} = If_{j,k}. \tag{13}$$

Consider now the interval $[f_{k-1}, a_k - L_k)$. This interval is meaningful only when $\beta_{k-1} = 0$, otherwise it is void. Similarly to $\Gamma f_{j,k}$, given a job J_k and a higher priority task τ_j, an array $\Gamma L_{j,k}$ of Boolean variables $\Gamma L_{j,k}[q] \in \mathbb{B}$ is defined to count the number of jobs of τ_j inside the interval. These jobs occur between f_{k-1} (thus they do not interfere with J_{k-1}) and the beginning of the busy period for J_k. Here, an active element $\Gamma L_{j,k}[q] = 1$ indicates that the q-th job of τ_j does not contribute to the busy period of J_k (it is before the start of its busy period). The size of the array $\Gamma L_{j,k}$ can be bounded, e.g., by $\left\lceil \frac{T_i - r_i + \mathcal{J}_j}{T_j} \right\rceil$, but again the effective number of jobs may be lower. Here, a value at 0 implies that the corresponding job of τ_j is already in the busy period for J_k. When this happens, all the following values must also be 0. The corresponding constraint then follows.

	$j=1$			$j=2$		
	$k=1$	$k=2$	$k=3$	$k=1$	$k=2$	$k=3$
$If_{j,k}$	3	4	7	1	1	2
$IL_{j,k}$	0	3	4	0	1	1
$\Delta_{j,k}$	3	1	3	1	0	1
$\Lambda_{j,k}$	0	0	0	0	0	0

Table 2. Counting higher priority (interfering) jobs for the example of Figure 1

Constraint 14. *If the q-th element of the array $\Gamma L_{j,k}$ corresponds to a job of τ_j that is already in the busy period for J_k, then all the following jobs do contribute to that busy period, i.e.:*

$$\forall j < i, \forall k, \quad \Gamma L_{j,k}[q] \geq \Gamma L_{j,k}[q+1]. \tag{14}$$

Constraint 14 is valid in both cases of $\beta_{k-1} = 0$ and $\beta_{k-1} = 1$. However, when $\beta_{k-1} = 1$ the interval $[f_{k-1}, a_k - L_k)$ is void and for all the elements of the array it must be $\Gamma L_{j,k}[q] = 0$. This is encoded in a constraint using the big-M formulation:

Constraint 15. *If $\beta_{k-1} = 1$ all variables $\Gamma L_{j,k}[q]$ are inactive, i.e.:*

$$\forall j < i, \forall k, \forall q, \quad \Gamma L_{j,k}[q] \leq M \cdot (1 - \beta_{k-1}). \tag{15}$$

Now, $\Gamma L_{j,k}$ can be leveraged to compute the total number of jobs of τ_j in the interval $[f_{k-1}, a_k - L_k)$ that are guaranteed not to interfere with J_{k-1} or J_k. We introduce the auxiliary variable $\Lambda_{j,k}$, counting the jobs of τ_j in $[f_{k-1}, a_k - L_k)$, which is defined as $\Lambda_{j,k} := \sum_q \Gamma L_{j,k}[q]$. The following constraint then holds.

Constraint 16. *Considering two successive intervals $[0, f_{k-1})$ and $[f_{k-1}, a_k - L_k)$ when $\beta_{k-1} = 0$, it holds that $If_{j,k-1} + \Lambda_{j,k} = IL_{j,k}$. This conditional constraint is coded as:*

$$\forall j < i, \forall k, \quad -M \cdot \beta_{k-1} \leq IL_{j,k} - If_{j,k-1} - \Lambda_{j,k} \leq M \cdot \beta_{k-1} \tag{16}$$

By combining Constraints 13 and 16, both variables $If_{j,k}$ and $IL_{j,k}$ are constrained to assume integer values only. A graphical visualization of all the variables counting the interfering tasks is shown in Figure 3, while a numerical example for the example of Figure 1 is shown in Table 2.

4.2.10 Refining the level-i idle time in a job window. Consider a job window $[a_k, a_{k+1})$ of J_k, if $\beta_k = 0$ the processor can be possibly idle only in the interval $[f_k, a_{k+1} - L_{k+1})$. We introduce the auxiliary variable Θ_k representing the total amount of higher priority workload in that interval, which can be computed as $\Theta_k := \sum_{j<i}(IL_{j,k+1} - If_{j,k}) \cdot C_j$. This variable can be leveraged to compute the level-i idle time as follows.

Constraint 17. *When $\beta_k = 0$, the idle time in the k-th job window can be computed as $\iota_k = a_{k+1} - L_{k+1} - f_k - \Theta_k$. This conditional constraint is coded as:*

$$\forall k, \quad -M \cdot \beta_k \leq \iota_k + \Theta_k - (a_{k+1} - L_{k+1} - f_k) \leq M \cdot \beta_k. \tag{17}$$

Trivially, for $\beta_k = 1$ the interval $[f_k, a_{k+1} - L_{k+1})$ is meaningless and the workload (thanks to Constraint 12) is $\Theta_k = \sum_{j<i}(If_{j,k} - If_{j,k}) \cdot C_j = 0$.

4.3 Objective Function

The goal of the analysis is to find one combination that gives the maximum number of deadline misses over K activations and compare it with the input constraint (m, K). The total number of

Type	Variables	Annotations
$\mathbb{R}$	f_k	Finish time of J_k with execution time C_i.
	$s_{j,k}$	Activation time of last job of τ_j before f_k.
	$If_{j,k}$	#jobs of τ_j within $[0, f_k)$.
	$IL_{j,k}$	#jobs of τ_j within $[0, a_k - L_k)$.
	$Is_{\hat{s},j,k}$	#jobs of τ_j within $[0, \hat{s})$, with $\hat{s} \in S_k$.
$\mathbb{B}$	$\Gamma f_{j,k}[\cdot]$	Where $\sum_p \Gamma f_{j,k}[p]$ is the #jobs of τ_j within: 1) $[a_k - L_k, f_k)$ when $k = 1$ or $\beta_{k-1} = 0$; 2) $[f_{k-1}, f_k)$ when $k > 1$ and $\beta_{k-1} = 1$.
	$\Gamma L_{j,k}[\cdot]$	Where $\sum_p \Gamma L_{j,k}[p]$ is #jobs of τ_j within $[f_{k-1}, a_k - L_k)$, if it exists.
	$\Gamma s_{\hat{s},j,k}[\cdot]$	Where $\sum_p \Gamma s_{\hat{s}}[p]$ is #jobs of τ_j within $[\hat{s}, f_k)$.

Table 3. Main variables defined for the job-continue strategy.

deadline misses of K consecutive jobs can be easily computed as $\mathsf{nM} := \sum_k b_k$. The objective function is then defined accordingly as follows:

$$m_{max} = \max(\mathsf{nM}), \tag{18}$$

where the value m_{max} is constrained inside the interval $[m+1, K]$. If a value exist in that range, the analysis ends and the (m, K) constraint is violated; otherwise, the constraint holds.

If no interval constraint $[m+1, K]$ is defined, the objective function (18) can be used to implement an optimization problem that checks an upperbound of the maximum number of deadline misses for K successive activations.

5 FORMULATION FOR JOB-CONTINUE STRATEGY

In this section, the MILP constraints that are specific for the weakly hard analysis of systems with a job-continue policy are presented. All the variables and constraints in this section complement the ones of Section 4 to provide the complete formulation.

5.1 MILP Variables and Scheduling Constraints

5.1.1 Bounding the finish time. According to the job-continue strategy, each job executes until completion. The finish time $f_k \in \mathbb{R}$ of each job J_k of τ_i, is constrained as follows.

Constraint 18. *The response time of a job J_k is bounded between r_i and R_i, i.e.,*

$$\forall k, \quad r_i \leq f_k - a_k \leq R_i. \tag{19}$$

For any two consecutive jobs of τ_i, the following precedence constraint holds.

Constraint 19. *The time interval between two consecutive finish times is bounded as follows:*

$$\forall k, \quad C_i \leq f_{k+1} - f_k \leq T_i + R_i + \mathcal{J}_i - r_i. \tag{20}$$

Proof. Since each job of τ_i executes for C_i time units, the lower bound trivially holds. The upper bound follows from the fact that the earliest completion time for J_k is when it is released without jitter and has its shortest response time r_i. The latest finish time for J_{k+1} is when it is released with maximum jitter and completes with its WCRT (i.e., $T_i + \mathcal{J}_i + R_i$ time units after a_k). □

5.1.2 Schedulability of J_k. The finish time f_k of each each job J_k must be compared with its (absolute) deadline d_k, computed as $d_k = L_1 + (k-1)T_i + D_i$, to check its schedulability. As introduced in Section 4.2.5, a Boolean variable b_k is used to model a deadline miss event. Considering the

job-continue strategy, the value of b_k can be defined as follows:

$$b_k = \begin{cases} 0 & \text{if } f_k \leq d_k; \\ 1 & \text{otherwise.} \end{cases}$$

Leveraging the big-M technique, the value of b_k is then encoded by the following linear constraint.

Constraint 20. *The variable b_k representing the event of missed deadline for J_k is encoded as:*

$$\forall k, \quad -M \cdot b_k \leq d_k - f_k < M \cdot (1 - b_k). \tag{21}$$

5.1.3 Computing f_k. The finish time f_k of a job J_k is computed using a relaxation of Equation (5), as presented in the following constraint.

Constraint 21. *The value of the finish time of J_k satisfies the following equality:*

$$\forall k, \quad \sum_{j<i} If_{j,k} \cdot C_j + k \cdot C_i + \sum_{k'<k} \iota_{k'} = f_k. \tag{22}$$

Indeed, Equation (22) allows for multiple solutions, where the exact response time is the lowest such solution f_k, and all the others are conservative upper bounds. Using such an equation is computationally efficient, but may result in a pessimistic estimate of f_k (and thus of the number of missed deadlines) since the optimizer is free to select any solution that satisfies Equation (22). Additional constraints are presented later in Section 5.2, when a more refined formulation is required, together with a more detailed discussion on the related issues and benefits.

5.1.4 Computing $a_k - L_k$. Similar to the computation of f_k, the time instant $a_k - L_k$, which represents the start of the busy period for J_k (when $\beta_k = 0$), can be formulated as the sum of multiple executions of interfering jobs and idle times. This is expressed by the following constraint.

Constraint 22. *The value of the time instant $a_k - L_k$ (which is meaningful only when $\beta_{k-1} = 0$) satisfies the following inequalities:*

$$\forall k, \quad -M \cdot \beta_{k-1} \leq \sum_{j<i} IL_{j,k} \cdot C_j + (k-1) \cdot C_i + \sum_{q<k} \iota_q - (a_k - L_k) \leq M \cdot \beta_{k-1}. \tag{23}$$

5.1.5 Busy period. If J_{k-1} does not interfere with the execution of J_k (i.e., $\beta_{k-1} = 0$), then $[a_k - L_k, f_k)$ is the k-th busy period. The total workload from higher priority tasks in $[a_k - L_k, f_k)$ is defined as $\Phi_k := \sum_{j<i}(If_{j,k} - IL_{j,k}) \cdot C_j$, and the relation $f_k - (a_k - L_k) = \Phi_k + C_i$ follows directly. The MILP formulation of the latter equation follows.

Constraint 23. *The busy period of the k-th job window, conditional to the case of $\beta_{k-1} = 0$, satisfies the following inequalities:*

$$\forall k > 1, \quad -M \cdot \beta_{k-1} \leq \Phi_k + C_i - f_k + a_k - L_k \leq M \cdot \beta_{k-1}. \tag{24}$$

For the case of $\beta_{k-1} = 1$ we consider the interval $[f_{k-1}, f_k)$ as the k-th busy period. The total amount of workload from higher priority tasks in the interval is then $\Phi'_k := \sum_{j<i}(If_{j,k} - If_{j,k-1}) \cdot C_j$, and it holds that $f_k - f_{k-1} = \Phi'_k + C_i$. As for the previous case, the MILP constraint follows.

Constraint 24. *The busy period of the k-th job window, conditional to the case of $\beta_{k-1} = 1$, satisfies the following inequalities:*

$$\forall k > 1, \quad -M \cdot (1 - \beta_{k-1}) \leq \Phi'_k + C_i - f_k + f_{k-1} \leq M \cdot (1 - \beta_{k-1}). \tag{25}$$

Given a value for β_{k-1}, only one between Constraints 23 and 24 can be active.

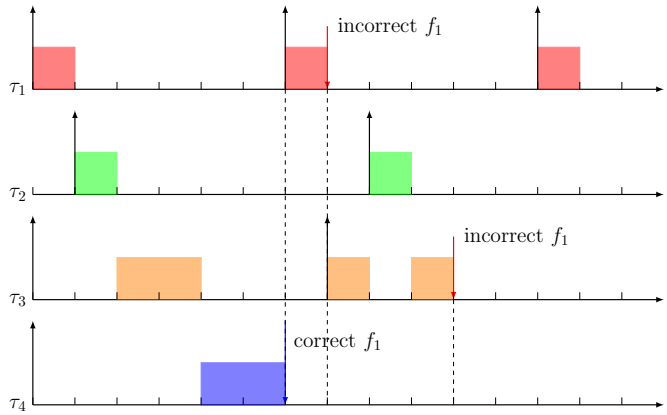

Fig. 4. An example of possible wrong values of f_1 for τ_4 obtained solving equation (22).

5.2 Additional Constraints: Refining the Formulation of the Finish Time

As previously introduced, formulating the minimization problem of (5) using only Constraint 21 can result in an excessively pessimistic evaluation for the weakly hard analysis. An example of what may happen is shown in Figure 4. Consider the computation of the finish time f_1 of the first job of task τ_4. The correct termination instant f_1 is indicated with a blue arrow. However, equation (22) is also satisfied by all the time instants indicated by the red arrows in the figure.

Unfortunately, since the weakly hard analysis is looking for the condition that produces the worst-case number of deadline misses in a problem window, the optimizer will select the *maximum* f_k value satisfying (22), if that value exceeds the deadline of J_k. Clearly, this introduces pessimism in the analysis, resulting in a larger worst-case number of deadline misses m. In order to limit this pessimism, and produce tighter (m, K) bounds, additional constraints must be defined.

Lehockzy et al. [21] provided an alternative test for schedulability. The test requires finding at least one time interval starting at the activation of the task under analysis τ_j and before its deadline such that the available time (the interval length) is greater than or equal to the time required for completing the execution of the task and all higher priority tasks. The endpoints of the intervals to be considered belong to a finite set and correspond to the activation instants of all higher priority tasks from the activation of τ_j to its deadline. These points are also the local minima of the function (5). It follows that the minimum value of f_k can be found by enforcing equation (22) and checking that there is no activation instant of any possible interfering task τ_j happening before f_k that satisfies the condition in [21].

Implementing this check would require the introduction of a high number of variables in the formulation, greatly impacting the runtime [41]. However, the results of the recent work in [27] show how a feasibility analysis performed on a carefully chosen small subset of interval endpoints still results in an accurate evaluation of the feasibility condition. Inspired by this result, we define a necessary but not sufficient condition for *non-schedulability*, obtained by considering only the *last* activation instant before f_k of each higher priority task. We achieve this by introducing an additional set of constraints that are leveraged for computing a much better approximation for the correct (minimum solution) value of f_k.

5.2.1 Constraints on the last interfering jobs for J_k. For each J_k and each higher priority task τ_j, the variable $s_{j,k}$ is introduced to define the activation instant of the last interfering job of τ_j, happening before f_k. In case there is no jitter, these activation instants can be properly computed as $s_{j,k} = (If_{j,k} - 1) \cdot T_j + \alpha_j$, while in the general case of $\mathcal{J}_j > 0$ each $s_{j,k}$ is constrained as follows.

Constraint 25. *The time interval between α_j and $s_{j,k}$ is bounded as:*

$$\forall j < i, \forall k, \quad (If_{j,k} - 1) \cdot T_j - \mathcal{J}_j \leq s_{j,k} - \alpha_j \leq (If_{j,k} - 1) \cdot T_j + \mathcal{J}_j. \tag{26}$$

The job of τ_j (with $j < i$) that starts at a time $s_{j,k} < f_k$ will finish before f_k due to priority ordering. This is enforced by the following constraint.

Constraint 26. *The interval between $s_{j,k}$ and f_k is bounded as:*

$$\forall j < i, \forall k, \quad r_j < f_k - s_{j,k} \leq T_j + \mathcal{J}_j. \tag{27}$$

Proof. As τ_j starting at $s_{j,k}$ must finish before f_k, then trivially $f_k - s_{j,k} > r_j$. On the other hand, if $f_k - s_{j,k} > T_j + \mathcal{J}_j$, this means that another activation of τ_j occurs before f_k, contradicting the definition of $s_{j,k}$, thus the bound follows. □

The set of all $s_{j,k}$ $\forall j < i$ for the k-th activation J_k is denoted as S_k. Given an element $\hat{s} \in S_k$, the number of job instances of τ_j within the interval $[0, \hat{s})$ is an integer number defined as $Is_{\hat{s},j,k}$. As for $If_{j,k}$ and $IL_{j,k}$, we relax $Is_{\hat{s},j,k}$ to real values, and the ceiling function is linearized as follows.

Constraint 27. *The number $Is_{\hat{s},j,k}$ of jobs of τ_j activated in the window $[0, \hat{s})$ is bounded as:*

$$\forall \hat{s}, \forall j < i, \forall k, \quad \frac{\hat{s} - \alpha_j - \mathcal{J}_j}{T_j} \leq Is_{\hat{s},j,k} < \frac{\hat{s} - \alpha_j + \mathcal{J}_j}{T_j} + 1. \tag{28}$$

Then, we define an array of Boolean variables $\Gamma s_{\hat{s},j,k}[p] \in \mathbb{B}$ that counts the number of jobs of τ_j inside the interval $[\hat{s}, f_k)$. An upper bound for the size of $\Gamma s_{\hat{s},j,k}$ is $\left\lceil \frac{T_i + \mathcal{J}_i}{T_j} \right\rceil$. Once again, the number of jobs effectively interfering may be lower than this upper bound. $\Gamma s_{\hat{s},j,k}[p] = 1$ indicates that the p-th job activation of τ_j in the specified time interval can interfere with the execution of J_k (the p-th job is activated before J_k completes, otherwise, $\Gamma s_{\hat{s},j,k}[p] = 0$). An element with 0 value implies that all the following elements are also valued at 0 (if a job does not interfere, neither the following jobs of the same task will do). This is encoded in the following constraint.

Constraint 28. *If the p-th element of the array $\Gamma s_{\hat{s},j,k}$ corresponds to a job of τ_j that does not interfere with J_k, then no later job can interfere with J_k, i.e.,*

$$\forall \hat{s}, \forall j < i, \forall k, \quad \Gamma s_{\hat{s},j,k}[p] \geq \Gamma s_{\hat{s},j,k}[p+1]. \tag{29}$$

The total number of activations of jobs of τ_j, interfering with the execution of J_k in $[\hat{s}, f_k)$ is then introduced as $\Delta s_{\hat{s},j,k} := \sum_p \Gamma s_{\hat{s},j,k}[p]$, and the constraint to enforce $Is_{\hat{s},j,k}$ follows.

Constraint 29. *Considering two successive intervals $[0, \hat{s})$ and $[\hat{s}, f_k)$, it holds that:*

$$\forall \hat{s}, \forall j < i, \forall k, \quad Is_{\hat{s},j,k} + \Delta s_{\hat{s},j,k} = If_{j,k}. \tag{30}$$

Finally, the non-schedulability condition for every time instant $\hat{s} \in S_k$ is obtained by enforcing that the time required for the execution of the task τ_i and all the interfering jobs is greater than the time available in the interval ending in $\hat{s}$.

Constraint 30. *The non schedulability condition for $\hat{s} \in S_k$ is enforced by the bound:*

$$\forall \hat{s}, \forall j < i, \forall k, \quad \sum_{j<i} Is_{\hat{s},j,k} \cdot C_j + k \cdot C_i + \sum_{k'<k} \iota_{k'} > \hat{s}. \tag{31}$$

Proof. First of all, note that the left side of inequality (31) is equivalent to the one of Equation (22) when computed for the point $\hat{s}$ instead of f_k. Since the correct value of f_k must be the *minimal* solution of Equation (22), this means that no other instants before f_k exist that satisfy that equation. Then, since by definition $\hat{s} < f_k$, $\forall \hat{s} \in S_k$, the inequality (31) must hold. □

Type	Variables	Annotations
$\mathbb{R}$	δ_k	Execution time of J_k.
	f_k	Finish time of J_k, with execution time δ_k.
	f_k^ε	Finish time of J_k, with execution time $\delta_k + \varepsilon$.
	$s_{j,k}^\varepsilon$	Activation time of last job of τ_j before f_k^ε.
	$If_{j,k}^\varepsilon$	#jobs of τ_j within $[0, f_k^\varepsilon)$.
	$Is_{\hat{s},j,k}^\varepsilon$	#jobs of τ_j within $[0, \hat{s})$, with $\hat{s} \in S_k^\varepsilon$.
$\mathbb{B}$	$\Gamma f_{j,k}^\varepsilon[\cdot]$	$\sum_p \Gamma f_{j,k}^\varepsilon[p]$ is #jobs of τ_j within: $[a_k - L_k, f_k^\varepsilon)$.
	$\Gamma s_{\hat{s},j,k}^\varepsilon[\cdot]$	$\sum_p \Gamma s_{\hat{s},j,k}^\varepsilon[p]$ is #jobs of τ_j within $[\hat{s}, f_k^\varepsilon)$.

Table 4. Main variables defined for the job-kill strategy.

Consider again the example of Figure 4. If one of the incorrect finish times marked with a red arrow is considered as a solution for f_k, then at least one value $\hat{s} \in S_k$ (namely, the activation times of the second jobs of τ_1 and τ_3 in the figure) exists such that inequality (31) does not hold. In this way, the two incorrect finish times marked in red are discarded.

5.2.2 Computational complexity. By checking only the points in S_k we can leverage a trade-off between precision and computational complexity of the MILP algorithm. This check prevents the pessimistic evaluation of f_k as a possible non-minimal solution of the fixed point response time equation (such as in the case presented in Figure 4). Our experiments show how the use of S_k is very effective in refining the bounds on the worst-case number of deadline misses, at the cost of a higher but still acceptable runtime for the weakly hard analysis. More details are provided in the experimental evaluation in Section 7.

6 FORMULATION FOR THE JOB-KILL STRATEGY

This section presents the weakly hard analysis under the job-kill strategy, i.e. when the execution of a job is immediately terminated when it misses its deadline. The immediate termination condition requires a set of different constraints, since the effective execution time of a task varies with respect to a possible deadline miss, i.e., a job that misses a deadline executes only partially. This means that only the *processed* execution before the deadline must be taken into account. In the following, the variables and constraints of the MILP formulation that are specific for the job-kill strategy are introduced. Again, these constraints complement the ones introduced in Section 4. In this analysis, we assume that all the tasks with priorities higher than τ_i always execute with their total execution time. Note that this a safe assumption with respect to the response time of τ_i (since some of them may be terminated early because of a missed deadline).

6.1 MILP Variables and Constraints

For the job-kill case a set of new variables are defined, summarized in Table 4. In the following, the relative constraints are presented.

6.1.1 Bounding the finish time. When using the job-kill strategy, the finish time of job J_k must be computed considering only the *processed* execution of J_k *before the deadline*. The amount of processed execution is defined as δ_k, where $0 \leq \delta_k \leq C_i$. A first constraint then follows.

Constraint 31. *The finish time of a job J_k under job-kill strategy is bounded by its deadline, i.e.:*

$$\forall k, \quad 0 \leq f_k - L_1 - (k-1) \cdot T_i \leq D_i. \tag{32}$$

Since $D_i \leq T_i$, no self-interference may happen when using the job-kill strategy. This means that L_k is always well-defined for all J_k. This is enforced by the following constraint.

Constraint 32. *Under job-kill strategy, $f_k \leq a_{k+1}$ always hold, thus it must hold that:*

$$\forall k, \quad \beta_k = 0 \tag{33}$$

6.1.2 Schedulability of J_k. The Boolean variable b_k introduced in Section 4.2.5 is used to evaluate the effect of deadline misses. By leveraging this variable, when the condition $b_k = 0$ holds (no deadline miss), the k-th job executes entirely, and the condition $\delta_k = C_i$ must be enforced. On the other hand, when $b_k = 1$, only partial execution occurs, thus $0 \leq \delta_k < C_i$. By introducing ε as an arbitrary small positive value, this is coded in MILP constraints as follows.

Constraint 33. *The bounds on δ_k, related to the value of b_k, are coded as:*

$$\forall k \quad (1 - b_k) \cdot C_i \leq \delta_k \leq C_i - \varepsilon \cdot b_k. \tag{34}$$

In order to guarantee that the correct value of δ_k is computed when partial execution occurs, we impose that if δ_k is increased by an arbitrarily small amount ε, the finish time cannot be less than the deadline. Note that, from a practical point of view, it is sufficient to consider ε equal to the time granularity of the system (tick). Another finish time variable f_k^ε is then introduced for each job J_k, and is computed considering an execution time of $(\delta_k + b_k \cdot \varepsilon)$. We refer to f_k^ε as the *extended finish time*. The value of f_k^ε is limited by the following constraints.

Constraint 34. *The interval between a_k and f_k^ε is bounded as:*

$$\forall k, \quad 0 \leq f_k^\varepsilon - a_k \leq R_i, \quad \text{and} \quad f_k^\varepsilon \geq f_k. \tag{35}$$

The introduction of f_k^ε is meaningful only when the k-th deadline is missed ($b_k = 1$). Otherwise, we force $f_k^\varepsilon = f_k$. In MILP notation, the following constraint for f_k^ε holds.

Constraint 35. *When $b_k = 1$, f_k^ε lies after the deadline, while when $b_k = 0$, f_k^ε lies within the deadline (since it is equal to f_k). This conditional constraint is coded as:*

$$\forall k, \quad -M \cdot b_k \leq L_1 + (k-1) \cdot T_i + D_i - f_k^\varepsilon < M \cdot (1 - b_k). \tag{36}$$

6.1.3 Number of interfering jobs from higher priority tasks. The number of higher priority jobs executing within the extended finish time f_k^ϵ can be computed using the same approach presented in Section 4.2.8. In particular, we define the number of jobs of τ_j in the interval $[0, f_k^\varepsilon)$ as a value $If_{j,k}^\varepsilon \in \mathbb{R}$ constrained by $\left\lceil \frac{f_k^\varepsilon - \alpha_j - \mathcal{J}_j}{T_j} \right\rceil \leq If_{j,k}^\varepsilon \leq \left\lceil \frac{f_k^\varepsilon - \alpha_j + \mathcal{J}_j}{T_j} \right\rceil$, and approximated as follows.

Constraint 36. *The number $If_{j,k}^\varepsilon$ of jobs of τ_j activated in the window $[0, f_k^\varepsilon)$ is bounded as:*

$$\forall j < i, \forall k, \quad \frac{f_k^\varepsilon - \alpha_j - \mathcal{J}_j}{T_j} \leq If_{j,k}^\varepsilon < \frac{f_k^\varepsilon - \alpha_j + \mathcal{J}_j}{T_j} + 1. \tag{37}$$

Since $f_k \leq f_k^\varepsilon$, the condition $If_{j,k} \leq If_{j,k}^\varepsilon$ always holds. Again, for computational efficiency, we have relaxed $If_{j,k}^\varepsilon$ to be a real value. We introduce an array of Boolean variables $\Gamma f_{j,k}^\varepsilon[p] \in \mathbb{B}$ that counts the number of jobs of τ_j inside the time interval $[a_k - L_k, f_k^\varepsilon)$. The value $\Gamma f_{j,k}^\varepsilon[p] = 1$ indicates that the p-th job of τ_j in the time interval interferes with J_k; otherwise, $\Gamma f_{j,k}[p] = 0$. A rough bound for its size is $\left\lceil \frac{T_i + R_i + \mathcal{J}_i}{T_j} \right\rceil$, but the number of jobs effectively contributing to the busy window of J_k may be lower. The propagation of 0 values is enforced in MILP formulation as follows.

Constraint 37. *If the p-th element of the array $\Gamma f_{j,k}^\varepsilon$ corresponds to a job of τ_j that does not interfere with J_k, then no later job can interfere with J_k, i.e.:*

$$\forall j < i, \forall k, \quad \Gamma f_{j,k}^\varepsilon[p] \geq \Gamma f_{j,k}^\varepsilon[p+1]. \tag{38}$$

The total number of activations of jobs of τ_j interfering with the execution of J_k in $[a_k - L_k, f_k^\varepsilon)$ is $\Delta_{j,k}^\varepsilon := \sum_p \Gamma f_{j,k}^\varepsilon[p]$. The following constraint is enforced to restore the integer values of $If_{j,k}^\varepsilon$.

Constraint 38. *Considering two successive intervals $[0, a_k - L_k)$ and $[a_k - L_k, f_k^\varepsilon)$, it holds that:*

$$\forall j < i, \forall k, \quad IL_{j,k} + \Delta_{j,k}^\varepsilon = If_{j,k}^\varepsilon. \tag{39}$$

Then, considering the interval $[f_{k-1}^\varepsilon, a_k - L_k)$, an array $\Gamma L_{j,k}^\varepsilon$ of Boolean variables $\Gamma L_{j,k}^\varepsilon[q]$ is introduced to count the jobs of τ_j in the interval. The bound for $\Gamma L_{j,k}$ is $\lceil \frac{T_i - D_i + \mathcal{J}_i}{T_j} \rceil$, and the propagation of 0 values is enforced as follows.

Constraint 39. *If the q-th element of the array $\Gamma L_{j,k}^\varepsilon$ corresponds to a job of τ_j that is already in the busy period for J_k, then all the following jobs do contribute to that busy period, i.e.:*

$$\forall j < i, \forall k, \quad \Gamma L_{j,k}^\varepsilon[q] \geq \Gamma L_{j,k}^\varepsilon[q+1] \tag{40}$$

The total number of jobs of τ_j in $[f_{k-1}^\varepsilon, a_k - L_k)$ is computed as $\Lambda_{j,k}^\varepsilon := \sum_q \Gamma L_{j,k}^\varepsilon[q]$. and is used to restore the integer values of $IL_{j,k}$ by means of the following constraint.

Constraint 40. *Considering two successive intervals $[0, f_{k-1}^\varepsilon)$ and $[f_{k-1}^\varepsilon, a_k - L_k)$, it holds that:*

$$\forall j < i, \forall k, \quad If_{j,k-1}^\varepsilon + \Lambda_{j,k}^\varepsilon = IL_{j,k}. \tag{41}$$

6.1.4 Computing f_k and f_k^ε. The finish time formulation for each job J_k can be computed similarly to the job-continue case presented in Section 5.1.3. The time interval $[0, f_k)$ consists of multiple busy periods and processor idle times, and the contribution of each job $J_{k'}$ with $k' \leq k$ is equal to its processed execution time $\delta_{k'}$. This can be efficiently coded as the following constraint.

Constraint 41. *The value of the finish time of J_k satisfies the following equality:*

$$\forall k, \quad \sum_{j<i} If_{j,k} \cdot C_j + \sum_{k' \leq k} \delta_{k'} + \sum_{k'<k} \iota_{k'} = f_k. \tag{42}$$

The term $\sum_{j<i} If_{j,k} \cdot C_j$ assumes that all higher priority tasks execute fully with C_j. This is a conservative assumption with respect to the weakly hard analysis of τ_i (increasing the interference cannot reduce the number of deadline misses suffered by τ_i).

A similar constraint is formulated for the extended finish time f_k^ε.

Constraint 42. *The value of the extended finish time of J_k satisfies the following equality:*

$$\forall k, \quad \sum_{j<i} If_{j,k}^\varepsilon \cdot C_j + \sum_{k' \leq k} \delta_{k'} + b_k \cdot \varepsilon + \sum_{k'<k} \iota_{k'} = f_k^\varepsilon. \tag{43}$$

The extra term $b_k \cdot \varepsilon$ only needs to be considered for the k-th step, as the effective execution time for all jobs with index $k' < k$ is equal to $\delta_{k'}$. As discussed in Section 5.1.3, the formulation of both f_k and f_k^ε without the minimization term is computationally efficient but may result in excessive pessimism. Additional constraints to overcome this issue are presented in Section 6.2.

6.1.5 Computing $a_k - L_k$. Similarly, the time instant $a_k - L_k$, which represents the start of the busy period for the job J_k and is always well-defined for the job-kill strategy, can be formulated as the sum of multiple interfering jobs and idle times. This is expressed as the following constraint.

Constraint 43. *The value of the time instant $a_k - L_k$ satisfies the following equality:*

$$\forall k, \quad \sum_{j<i} IL_{j,k} \cdot C_j + \sum_{k'<k} \delta_{k'} + \sum_{k'<k} \iota_{k'} = a_k - L_k. \tag{44}$$

6.1.6 Busy period for job J_k. Considering the finish time f_k, the total workload from higher priority tasks in the busy period $[a_k - L_k, f_k)$ is defined as $\Phi_k := \sum_{j<i}(If_{j,k} - IL_{j,k}) \cdot C_j$. The MILP constraint directly follows.

Constraint 44. *The busy period in the interval $[a_k - L_k, f_k)$ satisfies the following equality:*

$$\forall k, \quad \Phi_k + \delta_k = f_k - (a_k - L_k). \tag{45}$$

The same reasoning holds when considering f_k^ε and its corresponding busy period $[a_k - L_k, f_k^\varepsilon)$. By introducing the workload from higher priority tasks in that interval as $\Phi_k^\varepsilon := \sum_{j<i}(If_{j,k}^\varepsilon - IL_{j,k}) \cdot C_j$, the following constraint is enforced.

Constraint 45. *The busy period in the interval $[a_k - L_k, f_k^\varepsilon)$ satisfies the following equality:*

$$\forall k, \quad \Phi_k^\varepsilon + \delta_k + b_k \cdot \varepsilon = f_k^\varepsilon - (a_k - L_k). \tag{46}$$

6.2 Additional Constraints: Refining the Formulation of Extended Finish Time

As discussed for the job-continue case (see Section 5.2 for more details) additional constraints are required to reduce the pessimism in the computation of the finish time for the job-kill case. A set of constraints on the extended finish time f_k^ε are defined in the following. Since the variable f_k is upper bounded by the job deadline (in Equation (32)) and by the value of f_k^ε (in Eq. (35)), any feasible value for f_k^ε will ensure that also the upper bound f_k is lower than or equal to the deadline.

6.2.1 Constraints on the last interfering jobs for J_k. Consider the activation instant of the last interfering job of each higher priority task τ_j, before the extended finish time f_k^ε. These points in time are defined as $s_{j,k}^\varepsilon$. In case of no jitter, these activation instants can be computed as $s_{j,k}^\varepsilon = (If_{j,k}^\varepsilon - 1) \cdot T_j + \alpha_j$, and in the general case each $s_{j,k}^\varepsilon$ is constrained as follows.

Constraint 46. *The time interval between α_j and $s_{j,k}^\varepsilon$ is bounded as:*

$$\forall j < i, \forall k, \quad (If_{j,k}^\varepsilon - 1) \cdot T_j - \mathcal{J}_j \le s_{j,k}^\varepsilon - \alpha_j \le (If_{j,k}^\varepsilon - 1) \cdot T_j + \mathcal{J}_j. \tag{47}$$

The job of τ_j activated at $s_{j,k}^\varepsilon$ finishes before f_k due to priority ordering and is enforced as follows.

Constraint 47. *The time interval between $s_{j,k}^\varepsilon$ and f_k^ε is bounded as:*

$$\forall j < i, \forall k, \quad r_j < f_k^\varepsilon - s_{j,k}^\varepsilon \le T_j + \mathcal{J}_j. \tag{48}$$

The set of $s_{j,k}^\varepsilon \; \forall j < i$ is denoted as S_k^ε, and the number of job instances $Is_{\hat{s},j,k}^\varepsilon$ in the interval $[0, \hat{s})$, with $\hat{s} \in S_k^\varepsilon$, is constrained as follows.

Constraint 48. *The number $Is_{\hat{s},j,k}^\varepsilon$ of jobs of τ_j activated in the window $[0, \hat{s})$ is bounded as:*

$$\forall \hat{s}, \forall j < i, \forall k, \quad \frac{\hat{s} - \alpha_j - \mathcal{J}_j}{T_j} \le Is_{\hat{s},j,k}^\varepsilon < \frac{\hat{s} - \alpha_j + \mathcal{J}_j}{T_j} + 1. \tag{49}$$

Following the same approach as for the other cases, $Is_{\hat{s},j,k}^\varepsilon$ is relaxed as a real variable and an array of Boolean variables $\Gamma s_{\hat{s},j,k}^\varepsilon[p] \in \mathbb{B}$ is defined, with a rough upper bound of $\left\lceil \frac{T_i + \mathcal{J}_i}{T_j} \right\rceil$. Here, $\Gamma s_{\hat{s},j,k}^\varepsilon[p] = 1$ indicates that the p-th job activation of τ_j in $[\hat{s}, f_k^\varepsilon)$ interferes with the execution of J_k; otherwise, $\Gamma s_{\hat{s},j,k}^\varepsilon[p] = 0$. The following constraint guarantees the propagation of 0 values.

Constraint 49. *If the p-th element of the array $\Gamma s_{\hat{s},j,k}^\varepsilon$ corresponds to a job of τ_j that does not interfere with J_k, then no later job can interfere with J_k, i.e.,*

$$\forall \hat{s}, \forall j < i, \forall k, \quad \Gamma s_{\hat{s},j,k}^\varepsilon[p] \ge \Gamma s_{\hat{s},j,k}^\varepsilon[p+1]. \tag{50}$$

	Job-continue $(1,3)$			Job-continue $(2,5)$			Job-kill $(1,3)$			Job-kill $(2,5)$		
n	Av.(s)	Cnf	n/a	Av.(s)	Cnf	n/a	Av.(s)	Cnf	n/a	Av.(s)	Cnf	n/a
5	0.10	49.75%	0%	0.29	63.25%	0%	0.15	65%	0%	0.19	83%	0%
10	0.33	34.25%	0%	0.99	50.50%	0%	0.45	43.75%	0%	1.14	64%	0%
15	0.43	32.50%	0%	4.40	48%	0%	0.82	40.5%	0%	5.16	60.50%	0%
20	0.59	31.50%	0%	27.47	47.75%	0%	1.69	39.75%	0%	21.99	55.25%	0%
30	1.82	36.50%	0%	576.28	51.25%	7.25%	7.16	43%	0%	380.40	61.25%	4.25%

Table 5. Results using the **pessimistic** formulation for task sets with **random periods** and zero jitter. The table shows average runtime (Av.) expressed in seconds, percentage of confirmed (m,K) constraints (Cnf) and percentage of not completed runs (n/a).

The number of activations of jobs of τ_j that interfere with the execution of J_k in $[\hat{s}, f_k^\varepsilon)$ is defined as $\Delta s_{j,k}^\varepsilon := \sum_p \Gamma s_{\hat{s},j,k}^\varepsilon[p]$, and the discrete value of $Is_{\hat{s},j,k}^\varepsilon$ is enforced with the following constraint.

Constraint 50. *Considering two successive intervals* $[0,\hat{s})$ *and* $[\hat{s}, f_k^\varepsilon)$*, it holds that:*

$$\forall \hat{s}, \forall j < i, \forall k, \quad Is_{\hat{s},j,k}^\varepsilon + \Delta s_{j,k}^\varepsilon = If_{j,k}^\varepsilon. \tag{51}$$

Finally, a necessary condition for obtaining the minimal value of f_k^ε is that every time instant $\hat{s} \in S_k^\varepsilon$ is *not* a possible candidate finish time for J_k. This is obtained by enforcing that the time required for executing τ_i and all the interfering jobs is greater than the interval ending in $\hat{s}$.

Constraint 51. *The non schedulability condition for* $\hat{s}$ *is enforced with the bound:*

$$\forall \hat{s}, \forall k, \quad \sum_{j<i} Is_{\hat{s},j,k}^\varepsilon \cdot C_j + \sum_{k' \le k} \delta_{k'} + b_k \cdot \varepsilon + \sum_{k'<k} \iota_{k'} > \hat{s}. \tag{52}$$

7 EXPERIMENTS

In this section, the analysis proposed in the paper for the job-continue and job-kill policies is evaluated. For each strategy, the results of the application of the MILP weakly-hard analysis formulations are investigated. In detail, we consider **(i)** a *pessimistic formulation* that leverages Constraints 1-24 for the job-continue strategy, and Constraints 1-17, 31-45 for the job-kill strategy; and **(ii)** a *refined formulation*, which uses Constraints 1-30 for the continue strategy and Constraints 1-17, 31-51 for the kill strategy, respectively. For all formulations, we will use (18) as objective function.

Experimental setup. The tasks are generated with a variety of configurations, by varying the number of tasks ($n \in \{5, 10, 15, 20, 30\}$) and the total utilization factor ($U \in \{0.80, 0.85, 0.90, 0.95\}$). Overall, 100 different task sets for each combination of n and U have been created using the UUnifast algorithm [8], with periods randomly chosen in the interval $[10, 1000]$ and Rate Monotonic ordering of task priorities. Every task in the set has an implicit deadline. The first $n-1$ tasks have hard deadlines, while the lowest priority task τ_i has $WCRT_i > D_i$. This setup guarantees that all task sets in the experiment are not trivial (at least one deadline is missed by τ_i in the worst case). We present different tests where the value of jitter of all tasks is either set to zero, or uniformly selected in the range $0 \le \mathcal{J}_i \le 0.1 \cdot (D_i - C_i)$. When considering task sets with very high utilization, the weakly hard bounds can be easily violated by large jitters. A small amount of jitter may represent small time drifts and latencies in the task activations and corresponds to a *sensitivity test.*

To avoid biased results that can be possibly caused by random periods, we performed additional experiments by generating task sets with *pseudo-harmonic* periods, which are more representative of the period configurations found in real industrial applications. The period of each task τ_i in the set is chosen randomly such that it satisfies the following equation, $\forall i$:

$$T_i = 2^j \cdot 3^k \cdot 5^l \cdot 10, \quad \text{s.t.} \quad j+k+l \le 3, \quad \forall j,k,l \ge 0.$$

U	cont (1, 3)	cont (2, 5)	kill (1, 3)	kill (2, 5)
0.80	0.52 *s*	6.30 *s*	0.94 *s*	3.68 *s*
0.85	0.64 *s*	29.56 *s*	1.91 *s*	17.21 *s*
0.90	0.60 *s*	34.44 *s*	2.40 *s*	35.65 *s*
0.95	0.58 *s*	39.57 *s*	1.53 *s*	31.40 *s*

Table 6. Average runtime for the case of n=20 with **pessimistic** formulation, random periods and zero jitter.

	Job-continue (1, 3)			Job-continue (2, 5)			Job-kill (1, 3)			Job-kill (2, 5)		
n	Av.(s)	Cnf	n/a	Av.(s)	Cnf	n/a	Av.(s)	Cnf	n/a	Av.(s)	Cnf	n/a
5	0.15	16.75%	0%	0.42	24.50%	0%	0.18	48.75%	0%	0.28	59.75%	0%
10	0.46	7%	0%	1.04	16%	0%	0.28	64.75%	0%	0.51	71.50%	0%
15	0.58	3.50%	0%	4.26	12.25%	0%	0.29	74%	0%	0.59	74.75%	0%
20	0.74	1%	0%	23.95	8.25%	0%	0.41	74%	0%	1.09	74.50%	0%
30	1.43	1.75%	0%	215.40	7.25%	1.5%	0.54	75%	0%	0.92	75%	0%

Table 7. Results using the **pessimistic** formulation and **pseudo-harmonic** task sets with zero jitter: average runtime, percentage of confirmed (m, K) constraints and percentage of not completed runs.

Pseudo-harmonic tasksets have properties which are quite similar to the ones using harmonic periods, such as the characteristic of easily reaching high utilization factors without any deadline miss. Hence, we consider utilization factors of the set $U \in \{0.95, 0.97, 0.99\}$. The total number of tasks is varied as $n \in \{5, 10, 15, 20, 30\}$, defining 100 task sets for each combination of n and U.

The MILP weakly hard analysis is implemented in C++ using the CPLEX library[1]. A threshold of 3600 seconds for each CPLEX run has been set: if the weakly hard analysis for a task takes more than this limit, it is stopped and a timeout overrun is signaled together with the biggest value of m_{max} obtained up to that time. All tests have been performed on a machine with 128GB of memory and 40 cores of 2x Intel Xeon(R) CPU E5-2640 v4 running at 2.40 GHz.

7.1 Pessimistic MILP Formulation

7.1.1 Task sets with random periods. The first set of experiments applies to the weakly hard analysis formulated with the simplified response time analysis, for task sets generated with random periods and zero jitter. For every task set we tried $(m, K) = (1, 3)$ and $(2, 5)$, for both the job-continue and the job-kill strategy. The results are summarized in Table 5. The proposed MILP formulation shows remarkable timing performance. As expected, the runtime increases with both the number of tasks n and the size of the window K. In the worst case, the average time for completing the weakly hard analysis is a few minutes. The job-kill strategy results in a runtime which is mostly comparable with respect to the job-continue strategy. Results show how the job-kill strategy helps lower the number of missed deadlines, as indicated by the higher number of task sets that satisfy each (m,K) constraint in comparison with the job-continue case.

Table 6 shows the runtime distribution for the case $n = 20$ for different values of the utilization factor. The runtime of the weakly hard analysis depends on the processor utilization and the deadline miss strategy. For the job-continue case, the solution for $U = 0.95$ requires more computation time than the other cases, while for the job-kill strategy, a longer average runtime is observed for $U = 0.90$. A comparable behavior is found for all the other values of n.

Similar results are obtained for the same task sets with a random small jitter. The average runtime is basically unaffected and comparable with the case of no jitter, while the number of confirmed bounds decreases of nearly 19% on average for the continue and nearly 10% for the job-kill strategy.

[1]CPLEX is free for students and academics https://www.ibm.com/products/ilog-cplex-optimization-studio/resources

	Job-continue (1, 3)			Job-continue (2, 5)		
n	Av.(s)	Cnf	n/a	Av.(s)	Cnf	n/a
5	0.17	71.25% (↑21.5%)	0%	0.36	83.25% (↑20%)	0%
10	2.14	69% (↑34.75%)	0%	42.94	80% (↑29.5%)	1%
15	58.89	72 % (↑39.5%)	0%	984.66	72.5% (↑24.5%)	21.5%
20	831.22	68 % (↑31.5%)	13%	–	–	–
	Job-kill (1, 3)			Job-kill (2, 5)		
5	0.15	84.75% (↑19.75%)	0%	0.22	96.25% (↑13.25%)	0%
10	2.46	75.25% (↑31.5%)	0%	21.48	88% (↑24%)	0%
15	54.66	76% (↑35.5%)	0%	643.49	81.25% (↑26%)	10%
20	739.11	68 % (↑28.25%)	13%	–	–	–

Table 8. Results using the **refined** formulation for task sets with random periods and zero jitter: average runtime, percentage of confirmed (m, K) constraints and percentage of not completed runs. The difference of confirmed (m, K) constraints w.r.t. the ones obtained with the pessimistic formulation is shown in red.

U	cont (1, 3)	cont (2, 5)	kill (1, 3)	kill (2, 5)
0.80	100%	100%	100%	100%
0.85	98%	100%	99%	100%
0.90	67%	90%	74%	95%
0.95	11%	30%	28%	57%

Table 9. Percentage of verified (m, K) constraints for the case of $n = 10$ with the refined formulation.

7.1.2 Pseudo-harmonic task sets. The results obtained using pseudo-harmonic task sets with the pessimistic formulation are summarized in Table 7. A noticeable difference with respect to the ones obtained with task sets having random periods is that the percentage of confirmed constraints is way lower for the job-continue strategy, but higher for the job-kill strategy. This gap suggests that killing a job that misses a deadline is especially useful for restoring a correct behavior for task sets with pseudo-harmonic periods. Moreover, the runtime for the job-kill strategy is significantly shorter than the one for the job-continue policy, in particular for larger task sets.

7.2 Improved MILP formulation

7.2.1 Task sets with random periods. The refined MILP formulation presented in this paper improves the original one in [37], by providing a tighter bound on the maximum number of deadline misses. Our experiments show how the improvement is actually significant and worth the additional execution time in the optimization process. Table 8 shows the results, in which the improvement for the weakly hard analysis (with respect to results in Table 5, without the refinement) is highlighted in red. The additional constraints in the refined formulation help reduce the pessimism in the response time analysis, thus lowering the computed number of missed deadlines. This is supported by the fact that the percentage of confirmed (m,K) bounds increases in the average by 28.75% and 25.46% for different combinations of n and (m, K) values when the job-continue and the job-kill strategy are used, respectively. On the other hand, the refined MILP comes with a higher computational cost for the weakly hard analysis. As a consequence, although in most cases the runtime cost for the refined MILP is acceptable and does not exceed the timeout, the refined analysis does not scale to configurations with $n = 30$ or larger.

The improvements on the pessimistic analysis are not uniform across all possible utilization factors. As an example, consider the percentage of verified (m, K) constraints for the case of $n = 10$ and different values of U. Table 9 show the results. Figure 5 shows the difference, in percentage, of

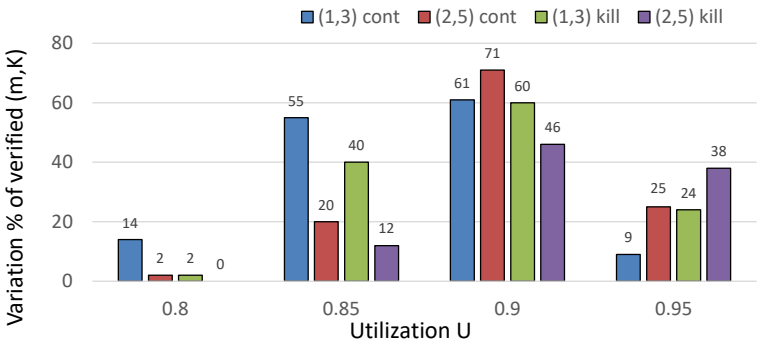

Fig. 5. Increment (in percentage) of verified (m, K) constraints for the case of $n = 10$ obtained with the refined formulation, with respect to the simple one.

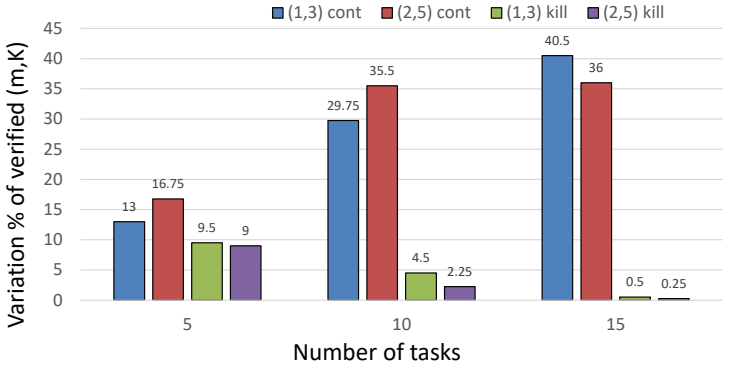

Fig. 6. Increment (in percentage) of verified (m, K) constraints for the case of $n = 5$, 10 and 15 for pseudo-harmonic tasksets, obtained with the refined formulation with respect to the simple one.

the verified task sets obtained with the refined formulation, with respect to the pessimistic one. The largest improvement is for $U = 0.9$ and $(m, K) = (2, 5)$, with the continue strategy, and the weakly hard schedulable sets increase by 71% when the refinement is used. Overall, given an (m, K) constraint, the advantage of the refinement in the weakly hard analysis is larger for larger sets and utilizations. When the total utilization is too high such as $U = 0.95$, it becomes difficult to validate almost any type of weakly hard constraint, as illustrated by the results in Table 9 and Figure 5. From Table 9, we also notice that the selection of the job-kill strategy instead of the continue is particularly effective for very high utilization values ($U = 0.95$).

Finally, we performed tests with random jitter, using the same values of the pessimistic analysis. The runtime results are comparable with the case of no jitter, while the number of confirmed bounds decreases more sharply (37,5% for the continue strategy and around 23% for the kill, on average), but still performing better than the results obtained with the pessimistic analysis for each combination (with an increase of 7% of accepted task sets on average).

7.2.2 Pseudo-harmonic task sets. In the pseudo-harmonic case, comparing the results obtained with the pessimistic formulation, though the runtime increases when using the refined formulation, this increment is negligible for the job-kill strategy, while being more sensitive when using the job-continue strategy. As a representative example, for $n = 15$ and $(m, K) = (2, 5)$ the average runtime is around 1277 seconds (with 17.75% of cases executing until the timeout) for the weakly hard analysis under job-continue strategy, and it is less than 4 seconds for the job-kill strategy. At the same time, as shown in Figure 6, the refined MILP weakly hard analysis helps boosting

the total percentage of verified (m, K) constraints when the continue-strategy is used. Since the simple MILP formulation performs already quite well for the job-kill strategy (Table 7), the further refinement does not have much room for improvement.

8 CONCLUSIONS

In this work, we propose an improved analysis for the weakly hard schedulability of offset-free, periodic task sets with fixed priority preemptive scheduling. The work targets analyzing weakly hard properties in the form of (m, K) constraints, by extracting a tight bound on the worst-case number of deadline misses that may happen every K iterations. The analysis is formulated using the MILP encoding for both the job-continue and the job-kill strategy to handle deadline misses. A set of additional constraints is produced, with the purpose of reducing the pessimism of the original approach presented in [37]. Runtime performance and comparison with the original formulation have been presented with extensive tests, showing that the improved analysis provides a good trade-off between complexity and precision.

REFERENCES

[1] Leonie Ahrendts, Sophie Quinton, Thomas Boroske, and Rolf Ernst. 2018. Verifying Weakly-Hard Real-Time Properties of Traffic Streams in Switched Networks. In *Real-Time Systems (ECRTS), 30th Euromicro Conference on*.

[2] Leonie Ahrendts, Sophie Quinton, and Rolf Ernst. 2017. Finite Ready Queues As a Mean for Overload Reduction in Weakly-hard Real-time Systems. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*. ACM, New York, NY, USA, 88–97. https://doi.org/10.1145/3139258.3139259

[3] Amir Aminifar, Petru Eles, Zebo Peng, and Anton Cervin. 2013. Control-quality driven design of cyber-physical systems with robustness guarantees. In *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 1093–1098.

[4] Amir Aminifar, Soheil Samii, Petru Eles, Zebo Peng, and Anton Cervin. 2012. Designing high-quality embedded control systems with guaranteed stability. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*. 283–292.

[5] Karl-Erik Årzén, Anton Cervin, Johan Eker, and Lui Sha. 2000. An introduction to control and scheduling co-design. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, Vol. 5. 4865–4870.

[6] Sanjoy Baruah and Alan Burns. 2006. Sustainable scheduling analysis. In *27th IEEE International Real-Time Systems Symposium (RTSS)*. 159–168.

[7] Guillem Bernat, Alan Burns, and Albert Liamosi. 2001. Weakly hard real-time systems. *Computers, IEEE Transactions on* 50, 4 (2001), 308–321.

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

[9] Reinder J Bril, Johan J Lukkien, and Rudolf H Mak. 2013. Best-case response times and jitter analysis of real-time tasks with arbitrary deadlines. In *Proceedings of the 21st International conference on Real-Time Networks and Systems*. ACM, 193–202.

[10] Tobias Bund and Frank Slomka. 2014. Controller platform co-design of networked control systems based on density functions. In *Proceedings of the 4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems*. ACM, 11–14.

[11] Tobias Bund and Frank Slomka. 2015. Worst-case performance validation of safety-critical control systems with dropped samples. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*. 319–326.

[12] Giorgio Buttazzo. 2011. *Hard real-time computing systems: predictable scheduling algorithms and applications*. Vol. 24. Springer Science & Business Media.

[13] Anton Cervin. 2005. Analysis of overrun strategies in periodic control tasks. In *Proc. 16th IFAC World Congress, Prague, Czech Republic*. Citeseer, 137.

[14] Goran Frehse, Arne Hamann, Sophie Quinton, and Matthias Woehrle. 2014. Formal analysis of timing effects on closed-loop properties of control software. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*. 53–62.

[15] W Geelen, Duarte Antunes, JPM Voeten, Ramon RH Schiffelers, and WPMH Heemels. 2016. The impact of deadline misses on the control performance of high-end motion control systems. *IEEE Transactions on Industrial Electronics* 63, 2 (2016), 1218–1229.

[16] Dip Goswami, Reinhard Schneider, and Samarjit Chakraborty. 2011. Co-design of cyber-physical systems via controllers with flexible delay constraints. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference*. IEEE Press, 225–230.

[17] Zain AH Hammadeh, Sophie Quinton, and Rolf Ernst. 2014. Extending typical worst-case analysis using response-time dependencies to bound deadline misses. In *Proc. of the 14th International Conference on Embedded Software*. ACM, 10.

[18] Chao Huang, Wenchao Li, and Qi Zhu. 2019. Formal verification of weakly-hard systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. ACM, 197–207.

[19] Mathai Joseph and Paritosh Pandya. 1986. Finding response times in a real-time system. *Comput. J.* 29, 5 (1986), 390–395.

[20] Pranaw Kumar and Lothar Thiele. 2012. Quantifying the effect of rare timing events with settling-time and overshoot. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*. 149–160.

[21] John Lehoczky, Lui Sha, and Yuqin Ding. 1989. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Real Time Systems Symposium, 1989., Proceedings*. IEEE, 166–171.

[22] John P Lehoczky. 1990. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *RTSS*, Vol. 90. 201–209.

[23] Joseph Y-T Leung and Jennifer Whitehead. 1982. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation* 2, 4 (1982), 237–250.

[24] Chung Laung Liu and James W Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)* 20, 1 (1973), 46–61.

[25] M. M Hamdaoui and P. Ramanathan. 1995. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. In *IEEE Transactions on Computers*.

[26] Luigi Palopoli, Luca Abeni, Giorgio Buttazzo, Fabio Conticelli, and Marco Di Natale. 2000. Real-time control system analysis: An integrated approach. In *Proceedings 21st IEEE Real-Time Systems Symposium*. IEEE, 131–140.

[27] Paolo Pazzaglia, Alessandro Biondi, and Marco Di Natale. 2019. Simple and General Methods for Fixed-Priority Schedulability in Optimization Problems. In *In Proceedings of the International Conference on Design, Automation & Test in Europe (DATE 2019)*.

[28] Paolo Pazzaglia, Claudio Mandrioli, Martina Maggio, and Anton Cervin. 2019. DMAC: Deadline-Miss Aware Control. In *31th Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[29] Paolo Pazzaglia, Luigi Pannocchi, Alessandro Biondi, and Marco Di Natale. 2018. Beyond the Weakly Hard Model: Measuring the Performance Cost of Deadline Misses. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[30] Sophie Quinton, Matthias Hanke, and Rolf Ernst. 2012. Formal analysis of sporadic overload in real-time systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 515–520.

[31] Parameswaran Ramanathan. 1999. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Transactions on Parallel and Distributed Systems* 10, 6 (1999), 549–559.

[32] Ola Redell and Martin Sanfridson. 2002. Exact best-case response time analysis of fixed priority scheduled tasks. In *Real-Time Systems, 2002. Proceedings. 14th Euromicro Conference on*. IEEE, 165–172.

[33] Lui Sha, Tarek Abdelzaher, Karl-Erik Årzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K Mok. 2004. Real time scheduling theory: A historical perspective. *Real-time systems* 28, 2-3 (2004), 101–155.

[34] Mahmoud Shirazi, Mehdi Kargahi, and Lothar Thiele. 2017. Resilient Scheduling of Energy-variable Weakly-hard Real-time Systems. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*. ACM, New York, NY, USA, 297–306. https://doi.org/10.1145/3139258.3139282

[35] Damoon Soudbakhsh, Linh TX Phan, Anuradha M Annaswamy, and Oleg Sokolsky. 2016. Co-design of arbitrated network control systems with overrun strategies. *IEEE Transactions on Control of Network Systems* (2016).

[36] Damoon Soudbakhsh, Linh TX Phan, Oleg Sokolsky, Insup Lee, and Anuradha Annaswamy. 2013. Co-design of control and platform with dropped signals. In *Cyber-Physical Systems (ICCPS), 2013 ACM/IEEE International Conference on*. 129–140.

[37] Youcheng Sun and Marco Di Natale. 2017. Weakly Hard Schedulability Analysis for Fixed Priority Scheduling of Periodic Real-Time Tasks. *ACM Trans. Embedded Comput. Syst.* 16, 5 (2017), 171:1–171:19.

[38] Wenbo Xu, Zain AH Hammadeh, Alexander Kroller, Rolf Ernst, and Sophie Quinton. 2015. Improved Deadline Miss Models for Real-Time Systems Using Typical Worst-Case Analysis. In *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*. IEEE, 247–256.

[39] Yang Xu, Karl-Erik Årzén, Enrico Bini, and Anton Cervin. 2014. Response time driven design of control systems. *IFAC Proceedings Volumes* 47, 3 (2014), 6098–6104.

[40] Yang Xu, Karl-Erik Årzén, Anton Cervin, Enrico Bini, and Bogdan Tanasa. 2015. Exploiting job response-time information in the co-design of real-time control systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2015 IEEE 21st International Conference on*. 247–256.

[41] H. Zeng and M. Di Natale. 2012. An efficient formulation of the real-time feasibility region for design optimization. *IEEE Trans. Comput.* 62, 4 (2012), 644–661.